Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Christian Håkon Torsten Inngjerdingen
Simen Aksland Møller

Incorporating Neighborhood Interactions and Vehicle Coordination in Bike Sharing Rebalancing with the X-PILOT Method

Master's thesis in Industrial Economics and Technology Management
Supervisor: Steffen J.S. Bakker
Co-supervisor: Henrik Andersson
June 2023

NTNU
Norwegian University of Science and Technology

Christian Håkon Torsten Inngjerdingen
Simen Aksland Møller

Incorporating Neighborhood Interactions and Vehicle Coordination in Bike Sharing Rebalancing with the X-PILOT Method

Master's thesis in Industrial Economics and Technology Management
Supervisor: Steffen J.S. Bakker
Co-supervisor: Henrik Andersson
June 2023

Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

NTNU
Norwegian University of
Science and Technology

NTNU
Kunnskap for en bedre verden

Department of Industrial Economics and Technology Management

TIØ4905 - Managerial Economics and Operations Research, Master's Thesis

Incorporating Neighborhood Interactions and Vehicle Coordination in Bike Sharing Rebalancing with the X-PILOT Method

Authors:
Christian Håkon Torsten Inngjerdingen
Simen Aksland Møller

Supervisor:
Steffen J.S. Bakker

Co-supervisor:
Henrik Andersson

8th June 2023

Preface

This thesis concludes our Master of Science at the Department of Industrial Economy and Technology Management at the Norwegian University of Science and Technology. It is written during the spring semester of 2023 and is a continuation of our specialization project in TIØ4500 Managerial Economics and Operations Research, from the fall 2022.

A special gratitude to our supervisors Associate Professor Steffen J.S. Bakker and Professor Henrik Andersson for valuable guidance and support throughout the project. We would also like to thank Urban Sharing and Urban Infrastructure Partners for providing helpful insights and access to necessary data.

Christian Håkon Torsten Inngjerdingen
Simen Aksland Møller

Trondheim, 8th June 2023

Abstract

This thesis examines the dynamics of neighborhood interactions and coordination of service vehicles within bike sharing systems (BSSs). Due to uneven distribution of demand across stations in BSSs, keeping the systems in balance is a significant challenge for operators. As stations become fully starved or congested, users may either abandon their attempts to pick up a bike or opt for neighboring stations, causing a spillover of demand. To ensure an efficient BSS, service vehicles are utilized to redistribute bikes between imbalanced stations. These rebalancing operations normally take place within a dynamic environment throughout the day, with uncertainty in customer demand causing stochasticity. To address the complexities of rebalancing under such conditions, this thesis formulates the Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions (DSBRPNI). A gap in existing research is identified, wherein the interactions between stations are frequently overlooked. Therefore, the main purpose of this thesis is to investigate the impacts of neighborhood interactions and to develop a solution method that incorporates such effects.

To solve this computationally challenging problem, we develop a metaheuristic called the Explorative Preferred Iterative LOokahead Technique (X-PILOT), which builds on the PILOT method by Voß et al. (2005). When applied to the DSBRPNI, our method can efficiently solve the problem while incorporating essential real-life considerations. X-PILOT seeks to avoid myopia by assessing the future implications of rebalancing decisions using lookahead techniques. Furthermore, the method features a novel construction algorithm that facilitates improved resource utilization and exploitation of synergies within a multi-vehicle system.

To evaluate the quality of our solution method accurately, it is implemented in a discrete-event simulator that emulates real BSSs. Historical data from Trondheim City Bike, Bergen City Bike, Oslo City Bike and Citi Bike New York is used to generate realistic test instances of varying sizes and demand patterns. We enhance the accuracy of the simulator by introducing a roaming module that determines whether users roam to neighboring stations when starved stations are encountered. Additionally, we propose a novel evaluation metric aimed at providing precise assessments of system performance by incorporating roaming. Results indicate that the new metric can provide significantly different interpretations compared to traditional metrics.

The computational study shows that the choice of branching width in X-PILOT significantly impacts solution quality. Furthermore, the study shows that longer time horizons lead to fewer failed events, highlighting the value of X-PILOT's lookahead feature. The solution method is proven to exhibit high computational performance, solving the New York instance consisting of over 900 stations and five service vehicles within few seconds.

Lastly, our solution method is compared to other benchmark policies. For all tested in-

stances, X-PILOT outperforms the alternatives in terms of service rates. Specifically, X-PILOT excels in coordination of vehicles. We demonstrate that the marginal benefit of additional vehicles is greater for X-PILOT, indicating that a more coordinated approach is achieved. The effect of incorporating neighborhood interactions into decision making is also evaluated. In small systems with a high degree of unassisted balance, the effect is negligible. However, in larger and more imbalanced systems, we show that the incorporation of neighborhood interactions results in fewer failed events.

Sammendrag

Denne oppgaven undersøker effekten av nabolagsinteraksjoner og koordinering av servicebiler i bysykkelsystemer (BSS). På grunn av ujevn fordeling av etterspørselen mellom stasjonene i BSS, er det en betydelig utfordring for operatører å opprettholde balanse i systemene. Når stasjonene blir helt tomme eller fulle, kan brukerne enten gi opp forsøket på å hente ut en sykkel eller dra til nabostasjoner, noe som overfører etterspørselen. For å sikre et effektivt BSS, brukes servicebiler til å reallokere sykler mellom ubalanserte stasjoner. Disse operasjonene gjøres vanligvis i løpet av dagen i et dynamisk miljø, der usikkerhet i kundens etterspørsel skaper stokastisitet. For å håndtere kompleksitetene ved rebalansering under slike forhold, formulerer vi det dynamiske stokastiske bysykkelrebalanseringsproblemet med nabolagsinteraksjoner (DSBRPNI). En mangel i eksisterende forskning er identifisert, der interaksjoner mellom stasjoner ofte blir oversett. Derfor er hovedformålet med denne oppgaven å undersøke konsekvensene av nabolagsinteraksjoner, og å utvikle en løsningsmetode som hensyntar slike effekter.

For å løse dette beregningstunge problemet utvikler vi en metaheuristisk metode kalt Explorative Preferred Iterative LOokahead Technique (X-PILOT), som bygger på PILOT-metoden av Voß et al. (2005). Anvendt på DSBRPNI kan vår metode effektivt løse problemet samtidig som viktige hensyn fra virkeligheten blir ivaretatt. X-PILOT unngår kortsiktighet ved å vurdere fremtidige implikasjoner av beslutninger ved bruk av lookahead-teknikker. Videre introduserer metoden en ny konstruksjonsalgoritme som muliggjør forbedret ressursbruk og utnyttelse av synergier i et system med flere servicebiler.

For å få en presis evaluering av kvaliteten på vår løsningsmetode, implementeres den i en diskret hendelsessimulator som etterligner virkelige BSS. Historiske data fra Trondheim Bysykkel, Bergen Bysykkel, Oslo Bysykkel og Citi Bike New York brukes til å generere realistiske testinstanser med varierende størrelser og etterspørselsmønstre. Vi forbedrer nøyaktigheten i simulatoren ved å introdusere en roaming-modul som avgjør om brukere drar til nabostasjoner når de ankommer tomme stasjoner. I tillegg foreslår vi en ny evalueringsmetode med mål om å gi presise vurderinger av systemets ytelse ved å hensynta roaming. Resultatene indikerer at den nye metoden kan gi betydelig ulike tolkninger sammenlignet med tradisjonelle metoder.

Beregningsstudien viser at valget av bredde-parameter i X-PILOT har betydelig innvirkning på løsningens kvalitet. Videre viser studien at lengre tidshorisonter fører til færre mislykkede hendelser, hvilket fremhever verdien av X-PILOT sine lookahead-funksjoner. Vi demonstrerer at løsningsmetoden har høy ytelse, og løser New York-instansen bestående av over 900 stasjoner og fem servicebiler på få sekunder.

Til slutt sammenlignes vår løsningsmetode med andre benchmark-metoder. X-PILOT presterer bedre enn de alternative metodene for alle instanser. Spesielt utmerker X-PILOT seg på koordinering av kjøretøy. Vi viser at marginalnytten med ekstra servicebiler er

større for X-PILOT, noe som indikerer en mer koordinert tilnærming. Effekten av å hensynta nabolagsinteraksjoner i beslutningstakingen blir også evaluert. I små systemer med en høy grad av balanse er effekten ubetydelig. Imidlertid viser vi at inkludering av nabolagsinteraksjoner fører til færre mislykkede hendelser i større og mer ubalanserte systemer, som i Oslo og New York.

Table of Contents

List of Figures

List of Tables

Abbreviations

Bike Sharing

BSS	Bike Sharing System

Operations Research

CGH	Column Generation Heuristic
DBRP	Dynamic Bicycle Rebalancing Problem
DSBRPNI	Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions
GP	Greedy Policy
GPNI	Greedy Policy with Neighborhood Interactions
IRP	Inventory Routing Problem
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Programming
PILOT	Preferred Iterative LOok ahead Technique
SBRP	Static Bicycle Rebalancing Problem
X-PILOT	Explorative Preferred Iterative LOokahead Technique

Chapter 1

Introduction

As the world faces a high rate of urbanization and population growth, cities are experiencing rising levels of private motorized traffic. This creates issues like environmental pollution and traffic congestion, fueling the need for sustainable and efficient transportation methods. Bike sharing systems (BSSs) have become a central component of many cities' public transportation systems, and are expected to play an increasing role in future urban mobility (Shui & Szeto, 2020). The European Parliament recently passed a resolution calling for a doubling of kilometres cycled in Europe by 2030, highlighting the importance of including cycling in the mobility ecosystem (Delli, 2023). Presently, there are almost 2,000 BSSs worldwide, with most having emerged within the last decade, and plans underway for hundreds of new systems (Russell Meddin, 2022).

The concept of a BSS involves a service provider making bicycles available for public use across a city. In a conventional station-based BSS, users retrieve a bike from a station, cycle to their desired destination, and secure the bike at a nearby station with an available lock. A BSS allows a pool of bicycles to be shared among many users, providing an adaptable and eco-friendly mobility solution for individual transport.

Although the flexibility of starting and ending a trip at different locations benefits the user, it poses significant challenges for operators. Uneven distribution of demand across stations leads to an imbalance in the system, resulting in stations eventually being either empty (starved) or completely full (congested), commonly referred to as *violations*. If no bikes are available, users are unable to start a trip, and if a station is full, bicycles cannot be returned. As a result, operators need to rebalance the system by moving bicycles between stations to meet customer demand. The task of deciding how to rebalance the system is referred to as the *Bicycle Rebalancing Problem*, which is addressed in this thesis.

Rebalancing accounts for the largest part of the operational expenses for BSSs and other micromobility operators (Büttner et al., 2011; Heineke et al., 2020), and is undoubtedly decisive for customer satisfaction. Thus, ensuring efficient rebalancing operations is of great economical value to BSS operators. Combined with the evolution and rapid expansion of BSSs in recent decades, this has lead to an increased focus on the rebalancing problem in Operations Research. As BSSs have become more complex, with an increasing number of stations and bikes, larger fleets of service vehicles are utilized to redistribute bikes between stations. This calls for an advanced decision support system to efficiently coordinate rebalancing efforts. With other micromobility alternatives entering the market, such as e-scooters, it is essential for BSS operators to maintain high availability to stay competitive.

Despite an increasing amount of research conducted on the rebalancing problem, it remains a significant challenge for operators, and there is considerable room for improvement (Shui & Szeto, 2020). One aspect that has yet to be thoroughly examined is the possibility for users to *roam* to a neighboring station in case of a congestion or starvation. This type of neighborhood interaction is known as *the spillover effect*. It is reasonable to assume that an empty station without any nearby alternatives is more problematic compared to a situation where users can walk a short distance to a nearby station and pick up a bicycle there. The same logic applies to users wanting to park at a congested station. Hence, focusing on stations independently during rebalancing can lead to suboptimal solutions in terms of total user satisfaction and utility in the system. Despite the potentially important role of roaming, this topic has received limited attention in scientific literature (Datner et al., 2019).

The purpose of this thesis is to take a holistic approach to the Dynamic Bicycle Rebalancing Problem, including the spillover effect between stations. The underlying rationale is that it is preferable to maximize the number of users whose demand is met within reasonable travel distance, rather than minimizing violations by examining stations independently. The first main contribution of this thesis is a solution method that incorporates demand interactions between neighboring stations. Our previous work (Inngjerdingen & Møller, 2022) showed that exact methods are too slow when solving this problem for larger BSS instances. Therefore, we propose a metaheuristic called Explorative Preferred Iterative LOokahead Technique (X-PILOT), which builds on the PILOT method developed by Voß et al. (2005). The heuristic is applied in an efficient solution method for the Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions (DS-BRPNI). Second, we introduce a novel way of simulating and evaluating BSSs, incorporating the spillover effect to enable more realistic results and thereby facilitating better decision making. Moreover, with the significant costs of rebalancing operations, efficient resource utilization is decisive. While coordination of service vehicles is commonly overlooked in existing methods, we propose an integrated approach that considers all vehicles simultaneously, enabling exploitation of synergies in a multi-vehicle setting.

The company Urban Sharing, who is currently providing BSS services in several European cities, has supplied real-world data from the BSSs *Oslo City Bike*, *Trondheim City Bike*, and *Bergen City Bike*. Data from *Citi Bike New York* is also used to test the X-PILOT implementation on large BSS instances.

This thesis begins with a brief study of the history and technical development of BSSs in Chapter 2, including a discussion of challenges faced by modern BSSs. In Chapter 3, a literature review is presented with a main focus on the Dynamic Bicycle Rebalancing Problem (DBRP). Chapter 4 provides insights to the main characteristics of the DS-BRPNI, before an example problem is presented and illustrated to provide further clarity. In Chapter 5, the X-PILOT solution method is presented. Further, Chapter 6 explains how a discrete event simulator is used to imitate BSS operations and evaluate different rebalancing policies. Chapter 7 details on how real-world data from Urban Sharing is utilized in the simulator. A computational study is presented in Chapter 8. The study begins with tuning and analysis of critical parameters before the computational performance of the solution method is investigated. The effects of considering vehicle coordination and neighborhood interactions are examined in Chapter 9. This chapter also includes a comparison between our solution method and other benchmark polices. Finally, Chapter 10 provides concluding remarks before Chapter 11 discusses possibilities for future research.

Chapter 2

Background

In 2006, there were merely 30 bike sharing systems (BSSs) worldwide. Since then, there has been a tremendous growth leading to almost 2,000 active systems today, with just short of 9 million bikes in operation (DeMaio et al., 2021; Russell Meddin, 2022). BSSs have evolved from early stages where bikes were painted and made available on the streets to be used for free, into a large commercial industry. Technological advancements have enabled new solutions such as electric bikes, geofencing technology for virtual stations and GPS-tracking of bike movements. While these rapid developments over the past decades have led to new opportunities, they also bring new challenges for operators to handle.

In Section 2.1, the bike sharing concept is described in general terms. Section 2.2 details the historical development of BSSs, while Section 2.3 examines emerging technologies and the future of BSSs. In Section 2.4, the BSS technology company Urban Sharing is briefly introduced. Important challenges related to BSSs are addressed in Section 2.5. Finally, Section 2.6 discusses the importance of considering the spillover effect. All sections in this chapter are based on previous work by the authors (Inngjerdingen & Møller, 2022).

2.1 Bike Sharing Concept

Although there can be variations in size, equipment, and operation, most BSSs consist of four main components: bicycles, stations, users, and operators. In these systems, users begin by picking up an available bicycle at a given station. After traveling to their destination, they lock the bicycle at a vacant lock at a nearby station. Typically, BSS operators offer a smartphone app that allows users to locate stations, check availability and unlock bikes.

As discussed in Chapter 1, imbalances within the system can result in stations becoming either completely full or empty, leaving users unable to start or complete their trips. These issues are known as *starvations* and *congestions*, collectively referred to as *violations*. Operators utilize light trucks, known as *service vehicles*, to move bicycles around and maintain a balanced system, reducing the occurrence of these violations. However, predicting future demand can be challenging for operators, who must determine which stations to visit and how many bikes to load and unload. Rebalancing efforts can be performed throughout the day, referred to as *dynamic rebalancing*, or at night time when there is no customer demand, referred to as *static rebalancing*.

The payment methods for bike sharing systems can differ depending on the operators and system. While some systems are free of charge, others have a periodic subscription fee, a deposit payment or a pay-per-use based on rental period. In modern systems, payment and subscriptions management is most commonly handled through apps. Alternatively, credit cards and smart cards can be used for payment (Vallez et al., 2021). Bike sharing is often used to bridge gaps in traditional public transportation, and is generally an affordable option compared to public transport and driving private cars.

Bike sharing services are provided by various types of actors, including local governments, transport agencies, advertising companies, for-profit and nonprofit groups (Shaheen et al., 2010). However, not all bike sharing systems are self-sufficient and they often require funding through methods such as advertising and public-private partnerships. A prominent funding source is a partnership between municipalities and advertising companies, such as JCDecaux and Clear Channel. Under this model, the advertising company provides the bike sharing service, and in return get the right to advertise on city street furniture and billboards (Shaheen et al., 2010).

Several cost drivers impact the operating firm of a BSS, such as capital costs, maintenance fees, and credit-card fees. However, relocating bikes is generally considered the most significant cost driver, accounting for up to 30% of operating costs in European systems, as shown in Figure 2.1 (Büttner et al., 2011). In a slightly different context, Heineke et al. (2020) calculate the costs per ride for a shared, free-floating e-scooter system. They estimate that relocation accounts for 40-50% of total costs. Despite the lack of physical stations, free-floating e-scooter systems share many of the same characteristics as station-based BSSs, including the need for rebalancing. Therefore, it is reasonable to assume that these figures provide a somewhat accurate breakdown of costs for a BSS.

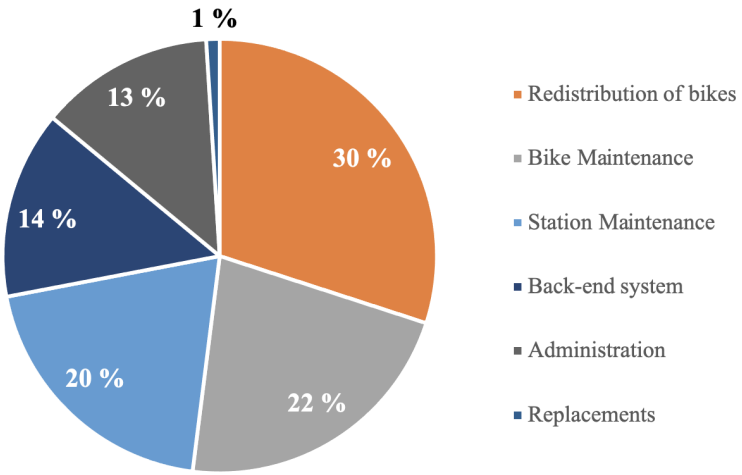


Figure 2.1: Operational costs of a BSS, based on Büttner et al. (2011).

2.2 Historic Development

Bike sharing systems have undergone significant evolution over time, not only in terms of the increasing number of bicycles and systems, but also in their design and the technologies used. The origins of bike sharing date back to Amsterdam in 1965 when Dutch inventor Luud Schimmelpennink encouraged people to paint their old bicycles white and leave

them anywhere in the city for others to use (Chandler, 2020). However, the initiative was primarily symbolic and a reaction from Schimmelpennink and his group of activists who aimed to unsettle the establishment. Predictably, the scheme failed due to the lack of a payment system and dedicated locks, making the bikes vulnerable to vandalism and theft. Furthermore, many of the bikes were confiscated by the police. Despite the "White bicycle plan" failing, it set the foundation for the BSSs that would follow.

Thirty years after the failure in Amsterdam, Schimmelpennink got a new opportunity when he was asked to help two designers in Copenhagen who set out to develop a new and better type of system. Copenhagen City Bike became the first large-scale bike sharing scheme in what has later been referred to as the second-generation systems. Initially, the system consisted of 1,100 bikes and 110 stations throughout the city center (Shaheen et al., 2010). To combat theft, the system used specially designed bicycles with parts that could not be installed on other bicycles. In addition, the bikes were locked to fixed docks. Users had to enter a coin deposit to unlock the bike, which was returned upon delivery, making the system free to use. The BSS was funded by commercial sponsors who used the bikes and racks for advertising. Copenhagen City Bike was a great success and was in operation until 2012, when it was replaced by a more modern system.

Despite its success, the Copenhagen bike sharing system still faced issues with theft and vandalism, which was likely caused by the anonymity of borrowing bikes. To address this challenge, Vélo à la Carte introduced a third-generation, IT-based system in Rennes in 1998 that required users to unlock bikes using personal smart cards (Shaheen et al., 2010). The system was still free to use, and also included other improvements, such as strategically placing bike stations near bus stations and park-and-ride facilities. With this approach, they aimed to integrate bike sharing into the community and provide seamless first and last-mile transport. Another example of a third-generation bike sharing system is Vélib' in Paris, which opened in 2007 and consisted of over 1,200 fixed stations and around 18,000 bicycles at its peak. Unlike the previously mentioned systems, Vélib' was not free to use and instead operated on a fee-based system, with additional costs after 30 minutes. Users could also buy 1-day pass, 1-week pass or 1-year subscription. It is one of the most successful large-scale BSSs in history, but was terminated and replaced in 2017.

Vélib' briefly held the title of the world's largest bike sharing system in terms of number of bicycles. However, as bike sharing became popular and established in Europe, countries in Asia also recognized its potential. Hangzhou was among the first Chinese cities to implement a bike sharing system with the launch of the Hangzhou Public Bicycle Service in 2008. Today, it is one of the oldest systems still in operation and one of the largest, consisting of more than 100,000 bicycles. China is now home to several of the largest bike sharing systems in the world, with 36% of all systems, according to DeMaio et al. (2021). The majority of these systems are based on docked third-generation systems, but new types, such as free-floating systems are growing rapidly.

Table 2.1 provides a rough summary of the development of bike sharing systems, categorized into different generations. While many current systems can be considered third-generation, new fourth-generation systems are constantly emerging. However, according to Chen et al. (2018), the recent rise of free-floating systems represents a transition to a new generation of bike sharing systems: fifth-generation systems featuring dockless bikes and advanced big data management capabilities.

Table 2.1: Characteristics of BSS generations

1st generation	2nd generation	3rd generation	4th generation
No payment	Coin deposit	Membership, free and fee-based	Membership, fee-based
Free-floating	Station based	Station based	Station based and free-floating
Painted bicycles	Specially designed bicycles	Specially designed bicycles	Electric and non-electric bicycles
Anonymous users	Anonymous users	Smart card	Mobile application
No tracking	No tracking	Pickup and drop-off tracking	Real time tracking of bicycles

2.3 New Kinds of Systems

Today, a wide range of different bike sharing systems are available, and new solutions continue to emerge in line with technological advancements. One system that has gained popularity is the free-floating system, where bicycles can be parked anywhere instead of at designated locks. The bikes are traced using GPS, and located by users through mobile applications. Despite their convenience for users, free-floating systems have been known to cause problems such as bikes being parked in inappropriate locations, disturbing pedestrians and other cyclists. In addition, the bicycles can have an unfortunate tendency to pile up in certain areas, and the early generations' problems of theft and vandalism have re-emerged (Chandler, 2020). However, solutions to these problems have been developed. After a massive influx of free-floating systems in China in 2017, local governments imposed regulations on the free-floating market in 2021, which reduced the number of available bikes. Additionally, improvements in geofencing technology have allowed for more precise management of bicycles in virtual parking hubs and no-parking zones, thus reducing the severity of previous problems (DeMaio et al., 2021).

Geofencing technology uses virtual boundaries and GPS or RFID to identify when a bicycle enters a specific geographic location. This technology is not limited to free-floating systems and can be used in station-based systems as well. The introduction of virtual stations has enabled the use of a higher number of stations - and hence, stations in closer proximity to one another. This reduces the inconvenience for users when roaming to a different station, and the severity of a starvation or congestion is largely determined by the state of nearby stations. Geofencing is also being used to increase the capacity of physical stations, by adding confined parking areas next to the docks. This has shown to reduce the problem of congestions and decrease the need for bike redistribution (Jiang & Jamba, 2019). These effects are changing the dynamics of BSSs and should presumably be taken into account in rebalancing decisions.

As previously mentioned, mobile applications are often used for payment and unlocking bicycles in modern BSSs. In systems equipped with "smart" stations, these apps can also contain real-time information on available bikes and locks. This allows users to easily check availability before travelling to a station, and consequently facilitating a more efficient transfer of demand between nearby stations.

2.4 Urban Sharing

Urban Sharing is a BSS software company, which develops and provides a technology platform used in several European BSSs. Among these are *Oslo City Bike*, the largest BSS in Norway with more than 2,800 mechanical bicycles. The system is dock-based, with over 250 physical stations (Russell Meddin, 2022). It opens at 05:00 in the morning, and is available until 01:00 at night. The BSS is a cooperation between the municipality of Oslo and the advertising company Clear Channel, in which the municipality offers advertising space in exchange for the system. Financing is based on user subscriptions, commercial revenue from advertising and sponsorship. Urban Sharing's solutions are also used for *Trondheim City Bike* and *Bergen City Bike*, which share similar characteristics to the BSS in Oslo. However, the systems are smaller in size with 66 and 100 stations respectively. Data provided by Urban Sharing for these three systems serve as a basis for the most of the example cases used in this thesis.

Urban Infrastructure Partner (UIP) is responsible for daily operations of the BSSs, including rebalancing, user support, and bike maintenance (Oslo City Bike, 2023). UIP is a sister company of Urban Sharing, and utilizes their software to support these operations. Urban Sharing's services include software for fleet management, rebalancing tools, demand predictions, maintenance reporting and logging, as well as bicycle and dock management. UIP use light trucks to perform dynamic rebalancing, i.e., rebalancing throughout the day. The operators responsible for rebalancing make use of data from Urban Sharing's platform to decide which stations to visit and how many bicycles to move. Such data can include a map with an overview of bike inventory for each station, demand predictions and details on bikes in need of maintenance. This operations software is illustrated in Figure 2.2.

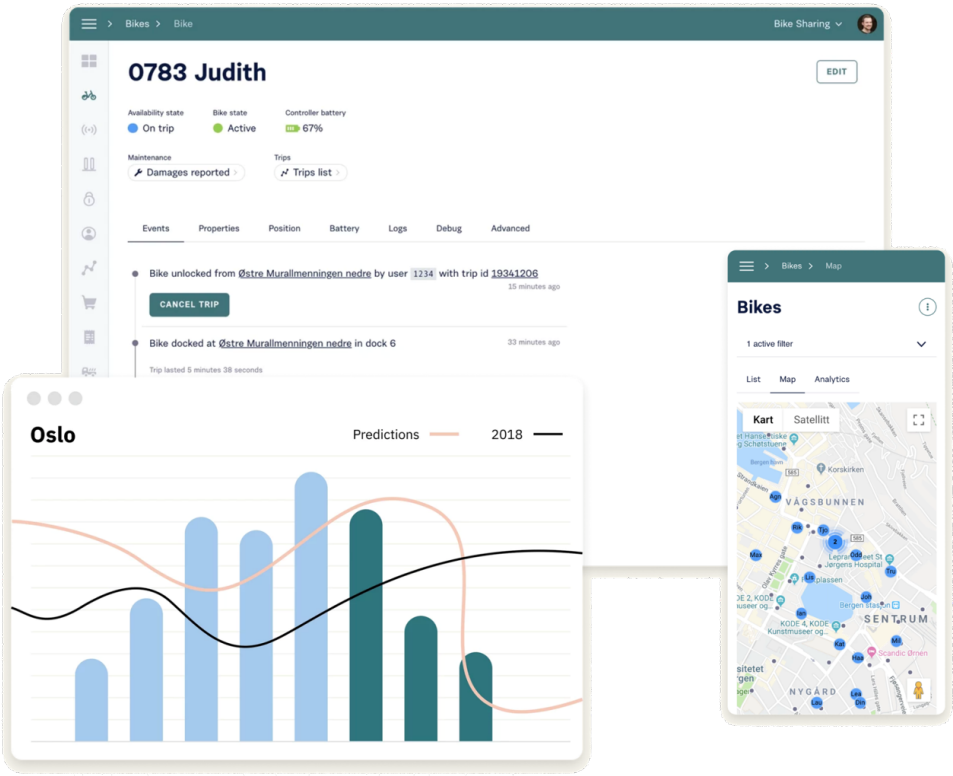


Figure 2.2: BSS operations software from Urban Sharing. Source: (Urban Sharing, 2023)

2.5 Challenges in Bike Sharing Systems

When implementing and operating a BSS, service providers must address multiple issues and make important decisions. These decisions include determining station density and location, station capacity, the number of bicycles per resident, and coverage area (Shui & Szeto, 2020). Additionally, to ensure smooth functioning of the system, there must be an adequate balance between the number of parking spaces and number of bicycles available in the system.

Moreover, the easy availability of bikes in public spaces presents several challenges. Among the most prominent are the problems of theft, vandalism and misplacement (Shaheen et al., 2010). This means that operators must replace lost bikes, as well as gather and repair broken ones. Early generations of BSS were especially prone to these issues, lacking technology to track and monitor bikes, and several systems were discontinued as a result (Shaheen et al., 2010). However, modern systems use GPS-tracking to monitor bike locations, and users can report issues directly to the operator through an app.

One of the most significant challenges still faced by BSS operators is keeping the system in balance. As shown in Figure 2.1, the rebalancing operations account for a significant part of the costs, and the availability of bikes and locks is naturally decisive for user satisfaction. In recent years, operations researchers have applied mathematical models in order to make rebalancing operations more efficient (Shui & Szeto, 2020). Such models provide decision support for operators, which can lead to better utilization of service vehicles and personnel. Even though mathematical models can aid in the daily operations, it is difficult to include all relevant real-life aspects in these models. Furthermore, the complexity of such models increases with the growth of BSSs, and with larger fleets of service vehicles needing to be coordinated across big cities.

2.6 The Spillover Effect

One real-life aspect that has yet received limited attention is neighborhood interactions, known as the *spillover effect*. When users looking for a bicycle arrive at an empty station, or observe this status in a mobile application, they have two options; either search for a bicycle at a neighboring station (referred to as roaming for bikes) or abandon the system in favor for other means of transportation (Datner et al., 2019). Therefore, an empty station can create a spillover of demand for bicycles to neighboring stations. Among other things, the willingness of individuals to walk to a neighboring station depends on the distance (Costa Affonso et al., 2021). Furthermore, when users want to return a bike at the end of a trip, they may find a full station. In docked systems, where bicycles must be placed in physical locks, this creates a challenge for the user who must locate an available lock at another station (referred to as roaming for locks). Therefore, a full station always transfers demand for locks to other stations.

Currently, most mathematical models only consider individual stations and disregard the fact that users can roam between stations to meet their demand. Neglecting these neighborhood interactions can lead to suboptimal decision-making, as noted by Datner et al. (2019). Similarly, Kloimüllner et al. (2014) argue that spillover of demand should be considered to make models more precise. Moreover, Hagen & Gleditsch (2018) argue that imbalanced stations are not necessarily problematic if nearby stations can handle the demand. They highlight the need for research investigating such interactions between

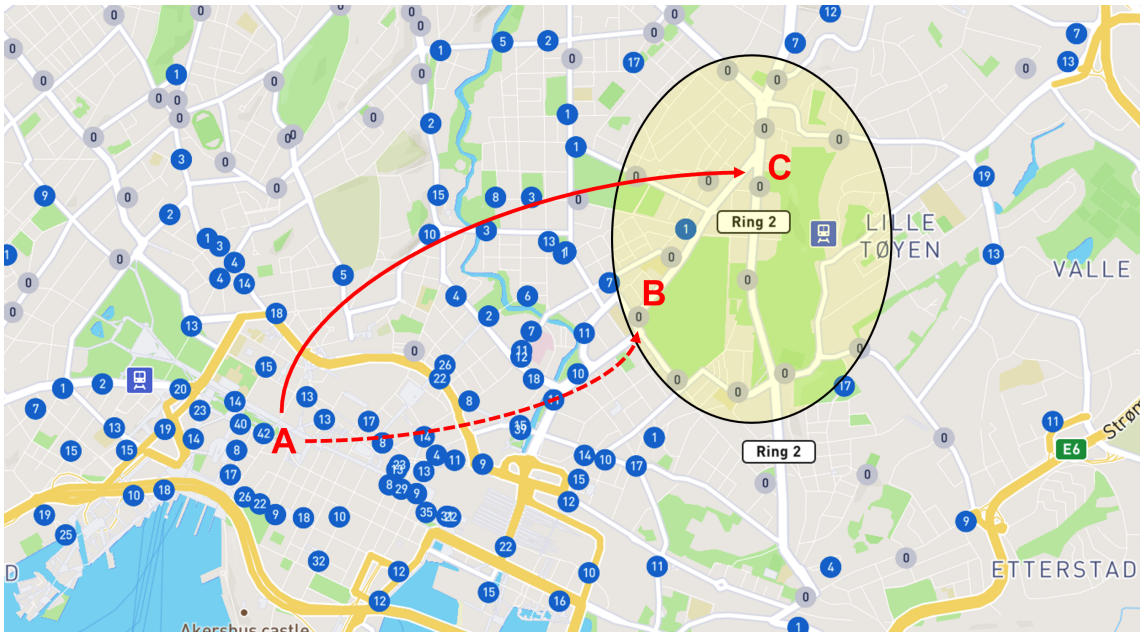


Figure 2.3: Snapshot of Oslo City bike, highlighting a zone with many empty stations. Two possible rebalancing moves are illustrated. Map adapted from Oslo City Bike (2023)

stations.

Consider the imbalanced system showed in Figure 2.3, which is a real snapshot from the Oslo BSS. The zone marked with a yellow oval suffers from many stations being completely empty. Suppose the operator wants to address this imbalance by redistributing a batch of bicycles from station A to a station in this zone. Most mathematical models consider the empty stations in isolation and choose to drop off the bikes at the station where the most violations can be avoided. When many models in addition punish the driving time between stations, bicycles are often being redistributed to the nearest empty station. Figure 2.3 demonstrates this approach with a rebalancing move from A to B. While this is beneficial for the users at B, it offers little value to the users at the other side of the zone. Furthermore, since B is situated on the border of the starved zone, it has several neighboring stations with available bicycles. Thus, rebalancing B does not appear to be the most advantageous move.

If spillover effects are taken into account by the model, the decision on where to redistribute the bicycles would be based on minimizing the total number of violations, including those from nearby stations. In Figure 2.3, this could result in the rebalancing move from A to C. The neighboring stations of C gain a benefit from the move since users looking for bicycles at these stations, now can roam to C. Since C has more empty neighbors compared to B, more stations benefit from the move to C. Users are likely to be more satisfied if they can walk a short distance to pick up bicycles at a nearby station, rather than having no bicycles available at all. Thus, taking spillover effects into account when deciding on the best rebalancing moves could lead to fewer severe violations and increased customer satisfaction.

Chapter 3

Literature Review

With the rapid growth and development of BSSs worldwide over the past few decades, the operations research community has shown increasing interest in addressing the challenges that arise. The topic includes complex decisions on several levels; from major strategic issues when designing a system, to everyday operations and maintenance. This literature review aims at giving an overview of the existing research literature, as well as an understanding of relevant modeling and solution approaches. Google Scholar has been used in search of relevant articles, applying keywords such as *BSS rebalancing problem, BSS neighboring stations rebalancing* and *BSS rebalancing zones*. Selection criteria have mainly been number of citations and publications in major operations research journals. Further, snowballing has been utilized to discover more relevant articles on selected topics.

In Section 3.1, the different planning levels of BSS problems are introduced, together with a brief discussion of their most common research topics. Section 3.2 discusses topics related to interactions between stations in the same geographical area. Section 3.3 looks at how the Dynamic Bicycle Rebalancing Problem can be categorized as an Inventory Routing Problem. Next, Section 3.4 considers applicable heuristic solution methods. Finally, relevant studies are compared in Section 3.5 before a conclusion and motivation of the thesis is presented in 3.6. Sections 3.1-3.3 and 3.5-3.6 are extensions of the literature review in Inngjerdingen & Møller (2022).

3.1 Planning Levels in Bike Sharing Systems

BSS decision problems can be categorized into three planning levels as shown in Figure 3.1 (Vogel, 2016). The strategic level focuses on high-level planning, including location and capacity of stations, number of bicycles and overall system design. The tactical level includes decisions regarding inventory levels at stations, as well as detection of broken bicycles and locks. Strategic level decisions serve as input to the tactical level decision making. Finally, the operational level relates to the daily operations. Based on input from the strategic and tactical levels, the daily operations aim to optimally redistribute bicycles between stations. They can also include pickup and repair of broken bicycles, as well as battery swaps for electrical BSSs. Higher levels serve as input for the lower levels, and the total performance of the system depends on all levels (Vogel, 2016). For instance, the placement of stations on a strategic level serve as foundation for the optimal number of bicycles chosen on a tactical level. This again affects the need for rebalancing efforts on an operational level.

Figure 3.1: The different planning levels of a BSS

3.1.1 Strategic Level

The overall system design is decided within the strategic level, where the decisions have a longer time horizon and typically involve capital investments. Business model and infrastructure investments are to be decided, which tend to have long-term impacts on the system performance. Such decisions are normally made prior to market entrance, and even when decisions can be changed later on, this usually involves costs (Vogel, 2016). According to Shui & Szeto (2020) there are three planning activities within the strategic level; bikeway design network, bicycle station design and fleet sizing design. The strategic level is not the main scope of this thesis, but a few relevant articles and decisions are presented to provide a context.

A central decision to be made on the strategic level is the location of stations, for which there are several different approaches in the literature. Using a Geographical Information Systems (GIS) approach, García-Palomares et al. (2012) suggest a model which takes into account statistical information regarding buildings, street networks, transport zones and public transportation networks. Methods of maximizing coverage and minimizing impedance are used to calculate an assumed distribution of demand. A set of obligatory and candidate stations are proposed, as well as the number of stations to employ. GIS can also be used to identify hot spots with insufficient bikes or bike racks (Wang et al., 2016). Lin & Yang (2011) propose a model for determining the number and locations of bicycle stations, taking into account perspectives of both users and investors. The model suggests a network structure of bike lanes connecting the stations and travel paths between each pair of origin and destination station.

External factors can be taken into account on a strategic level. Garcia-Gutierrez et al. (2014) account for the influence of co-existing transportation systems in station location. Romero et al. (2012) optimize the location of docking stations, using a model of private car and public transport modes as well as their interactions.

New technological developments are also influencing the strategic decision making. New bicycle types (e.g., electric, cargo and two-seated), free-floating systems and mobile bicycle docks each have their corresponding requirements for bicycle parking (Shui & Szeto, 2020). New research topics arise with this development, such as charging infrastructure for e-bikes, locations of virtual parking zones for free-floating bikes, or design of hybrid systems with different parking solutions and bicycle types.

3.1.2 Tactical Level

The goal of the problems at the tactical level is to optimize the utilization of the resources and the infrastructure of BSSs (Shui & Szeto, 2020). Thus, the results from the strategic problem, e.g., the number and locations of stations, can be seen as input for the tactical level. The problems on the tactical level have a relatively short time horizon, often a day at the time, but as the decisions are insensible to the real-time changes in the BSS, they are not considered operational. According to Shui & Szeto (2020), there are two main tactical problems in bicycle sharing service planning.

The first is known as the *Static Bicycle Rebalancing Problem* (SBRP). This problem is concerned with regulating the inventory levels of stations during the night, when the BSS is closed or demand is negligible. Relocation of bicycles is performed using one or several service vehicles and planning the vehicles' routes is an important aspect of these problems, in addition to determining the amount of bicycles to pick up and drop off. The SBRP is well-studied and many different techniques are being used to determine optimal rebalancing plans. Forma et al. (2015) utilize a three-stage mathematical programming based heuristic, where the first stage is a specialized heuristic that clusters stations based on geographic location and bicycle inventory. The second stage routes the repositioning vehicles and makes tentative inventory decisions. Finally, the last stage solves the original rebalancing problem with additional constraints. The last two stages are modeled as Mixed-Integer Linear Programs (MILPs) and solved by a commercial solver.

As the demand throughout the day is uncertain, it is important with thorough demand analysis and accurate forecast models. To include the future behaviour and demand patterns of a system, many researchers focus on determining the optimal initial inventory level of a station, i.e., how many bicycles and locks should be available at the beginning of the day. This is often the first part of the broader *Inventory Level Management Problem*, which is discussed further in Section 3.1.3. Espegren & Kristianslund (2016) use both expected demand and the variance to calculate a station's optimal initial inventory level. They suggest that the optimal level can be found when setting the probability of violation due to congestion equal to the probability of violation due to starvation. They argue that this makes the system well prepared for future demand. Similarly, Datner et al. (2019) also determine optimal initial inventory levels. However, this is done while considering spillover effects from nearby stations. In their model, users are allowed to roam between stations to rent and return bikes. Thus, an empty or full station can create a spillover of demand to nearby stations. Their objective function minimizes the sum of excess time, i.e., the difference between the total travel time and the ideal time, for all users. They show that the optimal inventory levels obtained by their method can save 7%-9% of excess user time in the system compared to models that do not consider the spillover effect.

The other main problem at the tactical level is the *Static Demand Management Problem*. Here, the aim is to increase cycling demand or motivate users to relocate bicycles. This can be achieved using incentives, such as different pricing for pairs of origins and destinations as studied in Haider et al. (2018). Another approach is for the operator to use parking reservation policies in order to regulate the demand over a given area or set of stations. Kaspi et al. (2016) explore the effect of allowing the operator to accept/deny a reservation of a bicycle lock, when the customer attempts to rent a bike.

3.1.3 Operational Level

The operational level focuses on short-term decisions taken throughout the day in response to real-time changes from daily activities. According to Shui & Szeto (2020), this includes *Inventory Level Management Problems, Dynamic Bicycle Rebalancing Problems (DBRPs)* and *Dynamic Demand Management Problems*. The DBRP is especially relevant for this thesis, and is further presented in sections 3.3 and 3.5.

Inventory Level Management Problems aim to determine the optimal target inventory level of usable bikes at each station *during* the opening hours of the BSS (Shui & Szeto, 2020). This differs from the optimal *initial* inventory levels discussed in Section 3.1.3, which serve as input for static overnight rebalancing. Often, the optimal inventory level is time dependent, meaning that a station should have different amounts of bicycles and locks available at different times throughout the day. Raviv & Kolka (2013) and Schuijbroek et al. (2017) model the Inventory Level Management Problem in a single station context, where inventory levels are determined independently and interactions with inventory levels of other stations are not considered. However, very often, interactions between inventory levels of different stations are not negligible, especially considering how spillover of demand from an empty station creates increased demand at nearby stations (Rudloff & Lackner, 2014).

Dynamic Demand Management Problems focus on incentivising users to create better resource utilization. They differ from Static Demand Management Problems by being performed throughout the day. Such demand management can be achieved either by introducing motivating incentives or by implementing regulations, in order to make users pick up and return bikes on certain stations (Shui & Szeto, 2020). The aim is to encourage, or require, pickups at stations with excess supply and deliveries at stations with low bike inventory levels. According to Shui & Szeto (2020), two of the most studied approaches in the literature are dynamic pricing incentives and parking space reservation.

Dynamic pricing is the most common demand management tool, and is a way of motivating user-based relocation in response to rapidly changing station inventory levels. Chemla et al. (2013) propose setting a price on return stations for each time interval, taking into account the extra walking and cycling effort required from a user selecting a non-preferred station. Ruch et al. (2014) use an agent-based model to show that dynamic price control can improve service rates, without the need to resort to redistribution staff. An integrated approach is suggested by Pfrommer et al. (2014), where dynamically varying rewards are offered to users based on current and predicted system state, combined with routing directions for redistribution staff. It is shown that a trade-off between user reward payouts and redistribution costs is possible to achieve, in order to minimize operating costs.

Two studies from Kaspi et al. (2014, 2016) analyze the performance of several different parking reservation policies. With such policies, the users must state their destination station upon bike rental, and a parking space is then reserved in the system. The studies conclude that policies with parking reservation outperform no-reservation policies in terms of total excess time of users in the system.

3.2 Neighborhood Interactions

As mentioned in Chapter 1, limited attention is given to the interactions between neighboring stations in the scientific literature. Still, some researchers attempt to quantify the

spillover effect and incorporate it into mathematical models. Rudloff & Lackner (2014) study the influence of weather conditions and full and empty neighboring stations when predicting the demand for bikes and locks. This new forecasting model is measured against historic demand, and the results show an improvement in predictions. Faghih-Imani & Eluru (2016) look at the spatial and temporal interactions between stations' departure and arrival rates. They show that part of the demand for bicycles/locks from a given empty/full station can be distributed to neighboring stations that are not empty/full. According to the authors, neglecting the presence of such effects, can result in biased model estimates.

There are few examples of models that include the spillover effect when determining static rebalancing plans. However, both Datner et al. (2019) and Costa Affonso et al. (2021) present algorithms that calculate the optimal initial inventory level on each station, considering spillover effects. Their objectives are respectively to minimize journey dissatisfaction (excess travel time for the users) and lost customer demand (users looking for bikes at empty stations). In addition to calculating the optimal initial inventory levels, Costa Affonso et al. (2021) use these results to determine the optimal route and loading/unloading decisions for the operator. It is worth noting that the authors make several significant simplifications, such as ignoring the demand for docking bicycles and using a predefined list of supplier stations. Still, both articles display the importance of including the interactions between stations. This is especially important when considering the trade-off between setting up many small stations or fewer stations with greater capacity, as the effects are larger in the former (Datner et al., 2019).

Finally, Regue & Recker (2014) propose a framework where they attempt to solve the Dynamic Bicycle Rebalancing Problem. One of the four core models in the framework is a vehicle-routing model that takes into account the state of the neighboring stations when evaluating the utility of visiting a station. The model maximizes the combination of utility gained by visiting a station with large inefficiency and the utility gained by visiting a station with a neighborhood of stations that is expected to have inefficiencies in future time steps. The authors define the neighborhood of a station as all stations that are located within 800 metres from the current station, but do not otherwise including roaming distances. The aim is to route the service vehicle to an area that is expected to have inefficiencies in future time steps. However, the model does not distinguish between starved and congested stations.

3.3 The Dynamic Bicycle Rebalancing Problem as an Inventory Routing Problem

The DBRP aims to redistribute bicycles between stations throughout the day, in order to maintain balanced inventory at stations. In general, the objective is to minimize unmet demand, in terms of bicycle rentals and returns. Due to bicycle inventory decisions, the DBRP can be considered as a variant of the Inventory Routing Problem (IRP).

In an IRP, inventory management decisions are integrated into the vehicle routing decisions. Usually, a supplier has to deliver products to a number of geographically dispersed customers in order to meet their demand (Coelho et al., 2014). In the DBRP, the operator can be seen as the supplier, while the stations correspond to the customers. The operator is responsible for transporting bicycles between stations and manage their inventory in order to meet customer demand. An extensive review of the literature regarding the IRP,

can be found in Coelho et al. (2014).

Table 3.1 presents the characteristics of the DBRP as an IRP using the classification framework described by Andersson et al. (2010). In the DBRP, a *finite* time horizon is considered, in which the state after the time horizon is dependent on the decisions made. Demand for bikes and locks experienced by the stations is *stochastic*. The topology of the DBRP can be described as *many-to-many* as there is no central facility, instead, bikes can be loaded and unloaded at any station. Routing is performed in a pickup and delivery setting, which is defined as *continuous* (Andersson et al., 2010). When demand is unmet, this is considered as *lost sales* as users can utilize other modes of transportation. Finally, both a *single* vehicle and *multiple*, *homogeneous* or *heterogeneous* vehicles, can be used in the DBRP.

Table 3.1: How the DBRP can be classified as an IRP. Based on the framework from Andersson et al. (2010)

Characteristic	**Alternatives**			
Time	Instant	Finite	Infinite	
Demand	Stochastic	Deterministic		
Topology	One-to-one	One-to-many	Many-to-many	
Routing	Direct	Multiple	Continuous	
Inventory	Fixed	Stock-out	Lost sales	Back-order
Fleet composition	Homogeneous	Heterogeneous		
Fleet size	Single	Multiple	Unconstrained	

3.4 Heuristic Solution Methods

The DBRP is a complex problem to solve, and operational use for rebalancing requires rather short solution times. For real-life instances, exact methods are often not applicable due to long solution times. A common approach to improve solution times for optimization problems, while still maintaining satisfactory solution quality, is to use heuristics. Several heuristic approaches to solving the DBRP, and problems with similar characteristics such as IRP and VRP, have been suggested in scientific literature. Table 3.2 provides a classification of some heuristic solution methods that are used for rebalancing problems, and use cases from scientific literature are briefly described in Section 3.4.1. An adapted Preferred Iterative LOok ahead Technique (PILOT) heuristic is used to solve the DBRP in this thesis. Literature on this method is presented in Section 3.4.2.

Table 3.2: Classification of heuristics that are used for rebalancing problems. The overview is non-exhaustive

Type	Local Search	Evolutionary Algorithms
Description	Iteratively improves an initial solution by searching nearby feasible solutions	Inspired by the process of natural selection. Iteratively evolves candidate solutions (population) by applying genetic operators
Examples	Simulated Annealing Variable Neighborhood Search Variable Neighborhood Descent Large Neighborhood Search	Genetic Algorithms Artificial Bee Colony Ant Colony Optimization Particle Swarm Optimization

3.4.1 Heuristics for the Bicycle Rebalancing Problem

Several heuristic solution methods have been applied to the Bicycle Rebalancing Problem (BRP). Di Gaspero et al. (2013a) use the local search approach Large Neighborhood Search (LNS) in order to obtain good solutions in a reasonable time. The LNS is coupled with constraint-based propagation to handle complex routing tasks. However, they are not able to outperform other benchmark models. Rainer-Harbach et al. (2013) apply two local search methods to solve a static rebalancing problem, namely Variable Neighborhood Search (VNS) and Variable Neighborhood Descent (VND). The VNS uses an embedded VND that exploits various specifically designed neighborhood structures. Experiments on benchmark istances indicate that high quality solutions can be found with this approach. The work is extended in Rainer-Harbach et al. (2015), by including a more sophisticated construction heuristic based on the PILOT method. A description of this method is found in Section 3.4.2. In addition, randomization is applied using Greedy Randomized Adaptive Search Procedure (GRASP). Results show that VNS performs particularly well on medium-sized instances, while a PILOT/GRASP-combination is best-performing on larger instances.

Ant Colony Optimization (ACO) is utilized as a search engine in Constraint Programming (CP) by Di Gaspero et al. (2013b). The routing variables are handled by the ACO, while the loading decisions are made by the CP. This approach is shown to outperform the pure CP formulation. Ma et al. (2021) propose eight genetic algorithms with various combinations of evolutionary mechanisms to solve the stochastic Static Bicycle Rebalancing Problem. Presented results demonstrate that the algorithms can effectively solve the Static Bicycle Rebalancing Problem across various areas, and with a higher solution efficiency than previous genetic algorithms. In order to reduce computation time, Szeto & Shui (2018) examine a novel set of loading strategies. The strategies are embedded into an Artificial Bee Colony algorithm to solve the SBRP. Szeto et al. (2016) apply Chemical Reaction Optimization (CRO) to solve a SBRP. The CRO handles vehicle routes, while a subroutine decides loading quantities. An enhanced CRO is used to improve the solution quality, and results show that the enhanced CRO provides high quality solutions within short computing times.

3.4.2 PILOT Method

The metaheuristic PILOT, as presented by Voß et al. (2005), is a tempered greedy method, which aims to create better solutions by avoiding the greedy trap. This is achieved by lookahead features, where possible future outcomes are evaluated for each greedy choice. Instead of measuring immediate gain, the PILOT method seeks to incorporate intelligent mechanisms to evaluate decisions based on the outcome after several iterations.

The PILOT method performs a known algorithm, such as a greedy construction heuristic, in a repetitive manner (Voß et al., 2005). A *master solution* is created by recursive calls of the algorithm. In each iteration, all possible successors from an incomplete master solution are evaluated. For each possible successor, a temporary complete solution is created. When these solutions are compared, combinations of moves are evaluated, rather than single moves. The best temporary solution is selected, and the corresponding successor is added to the master solution. This procedure is then repeated iteratively, until the master solution is completed. The mechanism aims to ensure that the solution which provides the highest immediate gain is not necessarily selected, but rather the solution that provides the most gain in total.

Rainer-Harbach et al. (2015) introduce the PILOT method as a construction heuristic for the Static Bicycle Rebalancing Problem. With a greedy heuristic as a basis, each candidate solution is extended by looking at potential successors. Rainer-Harbach et al. (2015) claim that the main issue with the greedy heuristic is that it always chooses a single, locally best successor. Consequently, a dense cluster of stations, which is further away than an isolated single station, may not be selected, even though it would yield a higher total benefit. The simple greedy algorithm does not recognize the cluster's overall value, and selects the single station. Rainer-Harbach et al. (2015) state that the PILOT method also incorporates future gains by visiting further stations in corresponding recursive calls. Hence, the PILOT method should be well fitted to a more complex rebalancing setting where neighborhood interactions are considered. Finally, Kloimüllner et al. (2014) adapt the PILOT method to the Dynamic Bicycle Rebalancing Problem. This approach is further discussed in 3.5.

3.5 Studies on the Dynamic Bicycle Rebalancing Problem

This section gives an overview and comparison of literature that studies the DBRP. Table 3.3 gives a summary of key findings from the examined literature. Through this section, research gaps in the literature are addressed, and used as foundation to motivate the topic of this thesis.

3.5.1 Objective Function

Most DBRP studies utilize similar objectives, namely minimizing unrealized demand and minimizing the corresponding cost. However, the exact specifications often differ and the problem is tackled with a variety of different objective functions.

Gleditsch et al. (2022) and Kloimüllner et al. (2014) minimize a weighted sum of violations and deviations from target inventory levels in the system. Deviations are punished in order to make the models less myopic. In addition, Gleditsch et al. (2022) reward vehicles starting on a trip that ends after the current planning horizon. This is done to reduce the idle time of vehicles. With a similar idea, Brinkmann et al. (2020) minimize the expected amount of unsatisfied demand. Transportation costs are neglected, based on a reasoning that vehicles and drivers are paid for either way from an operational view. Contardo et al. (2012) also minimize unmet demand, i.e., violations.

With a somewhat different approach, Ghosh et al. (2017) consider the objective a trade-off between minimizing lost demand (or maximizing profit) and minimizing travel cost incurred by vehicles. A dollar value is employed to both quantities, which are then combined into the overall profit. Similarly, Chiariotti et al. (2018) and Zhang et al. (2017) minimize user dissatisfaction related to respectively the probability of service failures and expected failures, while maintaining rebalancing costs as low as possible.

Regue & Recker (2014) is the only article among these to incorporate the status of neighborhood stations. The objective function maximizes utility from three elements; visiting a station with large inefficiency, visiting a station with a neighborhood that is expected to have large inefficiencies in future time steps, and minimizing travel time to intermediate pickup/delivery stations. The inefficiency of a station is measured in how many bikes that need to be added or removed in order to avoid violations.

3.5.2 Demand

Demand predictions in studies are primarily based on historical data. Brinkmann et al. (2020), Gleditsch et al. (2022) and Ghosh et al. (2017) all use historical data to simulate future demand over a predefined horizon. Kloimüllner et al. (2014) also use historical data. However, based on an hourly discretization of data, they construct piecewise linear demand functions which are later used to define time segments with similar demand behaviour. Chiariotti et al. (2018) model demand as Poisson variables, where birth and death rates are independent of current state. Further, Regue & Recker (2014) present a demand forecasting model based on a prediction method called Gradient Boosting Machines (GBM), which utilizes supervised regression-based machine learning. Contardo et al. (2012) introduce a stochastic demand function, only dependent on time.

3.5.3 Coordination of Service Vehicles

Solution methods can either be restricted to single-vehicle usage, or support multi-vehicle decision making. Even though several solution methods support multi-vehicle usage, the descriptions of vehicle coordination are often rather vague and ambiguous. Some solution methods implement simple handling of multi-vehicle instances with very limited coordination. Zhang et al. (2017) iteratively assign routes to one vehicle at a time, and lock the decisions. Similarly, Kloimüllner et al. (2014) present a greedy method and a PILOT method which both iteratively creates a tour for each service vehicle. However, some coordination is utilized here; after a vehicle tour is locked, the station inventory levels are updated based on the vehicle's actions, before creating the remaining vehicle tours. An alternative approach, using linear programming is also presented. Here, the order of the vehicles' station visits is taken into account, although this is only used to calculate loading quantities. There is limited coordination between the vehicles when constructing routes, which means that the model risks creating suboptimal routes for instances with multiple vehicles.

A common way to handle multiple vehicles is to divide the BSS into smaller zones, where each vehicle is responsible for one zone. A similar approach is utilized by Chiariotti et al. (2018). They do not coordinate vehicles, but rather assume that vehicles visit disjoint subsets of stations. Regue & Recker (2014) make a separation into two subproblems. The first subproblem identifies future loading and unloading quantities, while the second subproblem deals with assigning jobs to service vehicles through a vehicle routing problem. In Contardo et al. (2012), multiple vehicles are integrated into a space-time network, but coordination efforts are not discussed.

To the extent of our knowledge, only Gleditsch et al. (2022) and Brinkmann et al. (2020) provide coordinated approaches for multiple vehicles. Gleditsch et al. (2022) explicitly model the future routing and rebalancing decisions of all vehicles by means of mixed integer linear program. Brinkmann et al. (2020) utilize methods from approximate dynamic programming, however only considering operations at the current station and where to go next.

3.5.4 Modeling Characteristics

The literature in focus present several different approaches to modelling the DBRP. Gleditsch et al. (2022), Ghosh et al. (2017) and Brinkmann et al. (2020) model the problem as

a Markov Decision Process (MDP). An MDP is a mathematical model, used to represent a sequential decision-making process in situations where the outcomes of an action may be uncertain. Rewards and transition functions between states in the system are based solely on the current state and the current action (Puterman, 1990). The main objective of the decision maker is to determine a sequence of actions, that can optimize the system's performance over the decision making horizon. Zhang et al. (2017) and Contardo et al. (2012) model the problem as a space-time network. In Chiariotti et al. (2018), the state of each station is modeled as a Birth-Death Process.

The time aspect can either be handled in a continuous manner (Gleditsch et al., 2022; Kloimüllner et al., 2014), or using a discrete-time model in which a time horizon is discretized into shorter time periods (Contardo et al., 2012; Zhang et al., 2017). The discrete-time model of Contardo et al. (2012) considers states composed of initial position of vehicles, nodes for stations at different time periods, and a dummy node representing the end of a route in the planned schedule. A space-time network is presented, in which arcs represent trips between stations and time periods, or waiting at a station.

Most of the models consider starvations and congestions, which are typically included in station balance constraints. The congestion or starvation variables compensate for the unfulfilled demand, and there is no further handling of the events (Contardo et al., 2012; Gleditsch et al., 2022). This means that the bikes that arrive at a full station simply "disappear" from the system, instead of being transferred to a nearby station with available capacity. In addition, all demand at an empty station is lost, even if there are available bikes at nearby stations.

3.5.5 Solution Method

As discussed in Section 3.4, heuristic solution methods are commonly used to solve DBRPs due to computational complexity. Chiariotti et al. (2018) solve the problem using a greedy heuristic. Kloimüllner et al. (2014) examine several different approaches, including greedy and PILOT construction heuristics and VNS and GRASP for further improvement of constructed solutions. One way to solve the DBRP is by simplifying it, so it can be solved using a general solver. Regue & Recker (2014) propose reducing the size of the problem by solving it for one vehicle at a time, and only consider empty or full stations within a given distance of the current station. Zhang et al. (2017) reformulate a complex nonlinear optimization problem into an easier solvable MILP.

Gleditsch et al. (2022) break down the DBRP into subproblems considering shorter planning horizons. Further, they introduce a Column Generation Heuristic (CGH) to solve the subproblems in a rolling horizon fashion. Rolling horizon is an approach that involves dividing a large planning problem, into smaller, more manageable problems with shorter time frames. The plan is updated and revised as new information becomes available, allowing for increased flexibility and adaptability. This makes rolling horizon well suited for dynamic and stochastic problems. Contardo et al. (2012) first utilize Dantzig-Wolfe decomposition and solve the linear relaxation of the resulting problem using column generation, resulting in a lower bound. Secondly, Benders decomposition is applied to another formulation of the problem using information provided by the first solution, which provides an upper bound.

Table 3.3 summarizes the most important characteristics of the DBRP literature that has been reviewed in this section.

Table 3.3: Comparison of DBRP literature, including this thesis

	1 - Regue & Recker (2014)	**2 - Ghosh et al. (2017)**	**3 - Zhang et al. (2017)**	**4 - Chiariotti et al. (2018)**	**5 - Contardo et al. (2012)**	**6 - Brinkmann et al. (2020)**	**7 - Kloimüllner et al. (2014)**	**8 - Gleditsch et al. (2022)**	**9 - This thesis**
Problem focus	Demand forecasting, station inventory model, redistribution needs and vehicle routing	Rebalancing considering lost demand and cost of using vehicles	Dissatisfaction forecasting, bicycle repositioning and vehicle routing	When to redistribute bikes, optimal route	Dynamic rebalancing	Dynamic and stochastic rebalancing	Dynamic rebalancing, considering target fill levels and unsatisfied customers	Solving dynamic rebalancing sub-problems, addressing the problem of myopia	Solving dynamic rebalancing sub-problems, while considering the spillover effect
Objective function	Max. utility of visiting stations	Max. profit	Min. vehicle travel cost and expected user dissatisfaction	Min. user dissatisfaction and rebalancing cost	Min. unmet demand	Min. expected unsatisfied demand	Min. weighted sum of violations and deviations	Min. weighted sum of violations and deviations. Reward for starting trip	Max. utility
Modeling characteristics	ML, queuing theory, stochastic integer linear program	Markov decision process	Nonlinear model of a space-time network flow	Occupancy of station modeled as birth-death process	Arc-flow formulation on a space-time network	Markov decision process	Complete directed graph. Segments with monotonically increasing/decreasing demand	Markov decision process	Markov decision process
Solution method	Problem reduction and traditional solver	Lagrangian dual decomposition and station abstraction	Matheuristic	Greedy heuristic	Heuristic procedure based on Dantzig-Wolfe and Benders decomposition	Coordinated lookahead policy and value function approximation	Greedy and PILOT construction heuristics, VNS and GRASP	Decomposed into subproblems using rolling horizon. Column generating heuristic	Rolling horizon. Routes constructed using X-PILOT metaheuristic
Multiple vehicles	Yes, but limited coordination	Yes, but limited coordination	Yes, iteratively assigns a route to each vehicle	Assume vehicles visit different subsets of stations	Yes, but limited coordination	Yes, coordinated approach	Yes, but limited coordination	Yes, coordinated approach	Yes, coordinated approach
Demand	Demand forecasting using gradient boosting machines	Based on historical data, for each station and time step	Non-homogenous Poisson process	Poisson process, time varying rates based on historical demand	Stochastic demand function	Based on historical data	Derived from historical data, based on an hourly discretization	Based on historical data	Based on historical data. Routes are evaluated over random scenarios.
Spillover effects	Consider utility gained by neighboring stations	No	No	No	No	No	No	No	Yes

3.6 Conclusion and Motivation of the Thesis

Over the past decade, bike sharing systems have seen increased political focus as well as increasing attention from the academic community. Despite recent interest, the BSS literature is still limited, and not all relevant real-life aspects have been taken into account. There has been limited focus on the interactions between stations, which is the main area of focus in this thesis. Moreover, taking additional factors into consideration further raises the complexity of the rebalancing problem. This emphasizes the need for efficient heuristic solution methods that provide high quality results.

3.6.1 Considering Neighborhood Interactions

In existing literature, stations are predominantly considered separately with minimal focus to the interplay between them. Even though it has been pointed out that spillover effects and synergies between neighboring stations should not be neglected (Faghih-Imani & Eluru, 2016), this topic still represents a gap in the research literature.

As described in Section 3.2, there has been conducted some work that includes spillover effects in BSS problems. Rudloff & Lackner (2014) study the influence of full and empty neighboring stations when predicting demand for bikes and locks. Datner et al. (2019) and Costa Affonso et al. (2021) consider spillover effects when calculating a station's optimal state. Nonetheless, these studies are restricted to static systems, and only examine specific parts of the problem. They do not directly address the DBRP, which is the main problem of interest in this thesis.

Regue & Recker (2014) propose a way to include utility gained by imbalanced neighbors of a visited station, in order to take spillover-effects partly into account. However, the model defines all stations within 800 metres as neighbors, which are all treated equally. This means that the model neglects the difference between the utility gained by a station within a few metres distance, and a station 800 metres away. Furthermore, the model does not consider the actual roaming events, meaning the transfer of users between stations. In addition, there is no differentiation between starved and congested neighboring stations.

To the best of the authors' knowledge, the roaming effects in a DBRP have never been explicitly modeled and accounted for. In order to fill this gap, we develop a model that distinguishes between the severity of violations, depending on whether or not the user demand can be satisfied at a neighboring station. In addition, the severity is scaled based on how far the user actually has to move. Furthermore, congestions and starvations do not lead to bikes and demand simply "disappearing" from the system. Instead, in our approach demand is redistributed more realistically to other stations. The presented solution method aims to maximize user satisfaction on a more detailed level, distinguishing between dissatisfaction related to not having demand fulfilled at all, and dissatisfaction related to roaming to a nearby station.

3.6.2 Improving the Solution Method

In order to solve the Dynamic Bicycle Rebalancing Problem within a reasonable time, heuristic methods are commonly applied. Despite being susceptible to the issue of the greedy trap, the current literature frequently employs greedy methods. In this thesis, we design and implement a PILOT-inspired approach aiming at overcoming myopia by

looking ahead beyond the first step of rebalancing. Rather than evaluating solutions based on immediate gain, we estimate the total gain over a given future. The solution method utilizes computationally efficient greedy techniques that are iteratively called, enabling quick solution times.

Furthermore, we introduce a novel way of coordinating a fleet of vehicles. Most existing methods route vehicles independently, or assume that different subsets of stations are visited. The method presented in this thesis creates rebalancing plans in which all vehicles are included, rather than individual rebalancing routes for each vehicle. Rebalancing plans are constructed taken into account the location and state of other vehicles, as well as their possible future moves. This approach can enable utilization of synergies and help to prevent suboptimization.

Kloimüllner et al. (2014) and Rainer-Harbach et al. (2015) first introduced the PILOT method for bike rebalancing, however, there are some important differences in our approach. First, we incorporate spillover effects between stations, which is lacking in existing methods. Further, existing construction algorithms consider reduction in violations, deviation from target inventory levels and an urgency factor. We introduce a criticality score, which also incorporates the criticality of the neighborhood, time to violation and magnitude of demand.

Evaluation of solutions also differ from existing approaches. While current solutions only consider expected demand, we handle stochasticity by using scenarios. Each solution is evaluated over a range of possible future outcomes, allowing for more robust solutions to be selected. In addition, we introduce a discounting factor in the evaluation function that prioritizes station visits that are closer in time, and hence subject to less uncertainty.

Compared to existing PILOT methods, our approach, presented in Chapter 5, offers increased diversification. In current methods, branching to different solutions is only permitted in the initial step of the algorithm, before each solution is completed in a greedy fashion. Our version of the PILOT algorithm allows for branching in multiple steps, enabling a more extensive exploration of the solution space.

Chapter 4

The Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions

This chapter introduces the main characteristics of the Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions (DSBRPNI), which is the problem we aim to solve in this thesis. This is a variant of the more general Dynamic Bicycle Rebalancing Problem (DBRP) discussed in Chapter 3. First, a description of the real-world problem is provided in Section 4.1, before the problem is illustrated through an example in Section 4.2.

4.1 Problem Description

We consider a bike sharing system (BSS) consisting of a fixed number of physical stations spread across the city and a homogeneous fleet of bicycles. Each station has a fixed number of locks, limiting the number of bicycles that can be stored simultaneously. The system is accessible to users throughout the day, allowing them to pick up and drop off bicycles at different stations to meet their transportation needs. If a user arrives with a bicycle at a congested station, the user is unable to end the ride. This means that he must travel further and search for a station with available locks. However, in the case of a starvation, the user can either look for a bike at a nearby station or decide to use other means of travel.

4.1.1 Decisions to Be Made

To make the system available for as many users as possible, the service operators use service vehicles to continuously relocate bikes from stations with excess inventory to stations with an insufficient number of bikes. For this rebalancing operation, two main decisions must be made:

1. Which stations to visit
2. The number of bikes to load or unload at each station

Based on the taxonomy from Pillac et al. (2013), the real-world problem is both dynamic and stochastic. It is dynamic as the distribution of bicycles between stations is not known beforehand, but is continuously changing during the day. The problem is stochastic as future demands for bicycles and locks at stations are unknown, but assumed to follow a known distribution. Operators observe the state of the system as customer demand is realized throughout the day and perform actions based on this.

As the development of the system is both dependent on rebalancing decisions and the stochastic customer demand, the problem can be considered a Markov Decision Process (MDP). In MDPs, decisions can be made at discrete time points or when a specific condition is satisfied. In the DSBRPNI, a natural decision point for the operator is when a vehicle arrives a station. The problem can thus be considered an online optimization problem in which decisions are made in a sequential manner based on incomplete information. The problem can also be solved as an offline problem, in which a complete rebalancing plan is created beforehand. However, this would not allow operators to respond to actual realizations of demand. The offline problem is not further discussed in this thesis.

4.1.2 Available Information and Problem Assumptions

Labor and vehicle costs are naturally dependent on the number of vehicles utilized, but these are tactical and strategic decisions and are not part of the scope of this thesis. We assume that service vehicles and personnel are working continuously during their scheduled working hours. Under these assumptions, the related costs are not significantly impacted by the routing and loading decisions. Therefore, the costs of fuel and labor are neglected.

The problem can be solved for traditional stations with physical locks, virtual stations, or a combination of the two. However, the station capacity cannot change. Furthermore, only mechanical bikes are considered, meaning that there are no calculations of battery swaps or bike charging. Handling and repair of broken bikes is not included in the problem formulation.

When solving a rebalancing problem, the characteristics of the BSS are known. The BSS utilizes a certain number of service vehicles, with a given carrying capacity. The service vehicles can coordinate their routing and rebalancing decisions. In addition, service vehicles may be heterogeneous with different capacities. The vehicles drive between stations, and we assume driving times to be known and deterministic. Similarly, we assume that the time it takes for users to travel between stations, either by foot or on bike, is known. There is a handling time associated with loading and unloading of bikes, which is dependent on the number of bikes to be handled.

Each station has a certain number of locks, i.e., a given capacity. Furthermore, we assume that each station has defined an inventory level at any given time that is optimal for avoiding future violations. The calculation of this *target inventory level* is not discussed in this thesis. BSS operators can monitor the state of the system, which means that the number of bikes and locks available at each station is known at all times. This information can therefore be utilized when rebalancing decisions are made.

Future demand for bikes at stations is not known. However, historical arrive and leave intensities are known for all stations throughout the day, using an hourly discretization. This data can be used to generate expected values of the demand and stochastic scenarios.

4.1.3 Objective

Using the available information, the operator makes decisions to maximize the utility of the system, or in other terms, minimize the customer dissatisfaction. We assume the total customer dissatisfaction to be affected by two types of events. First, and most severe, is a starvation where the user is unable to pick up a bike, and is assumed to leave the system and use other modes of transportation. The second is related to a situation where the user is unable to pick up or deliver a bike at a given station, but roams to a neighboring station to do so. We assume dissatisfaction to increase with the distance of the roaming. Further, operators should avoid being too myopic. This can be achieved by performing rebalancing operations that minimize the stations' deviations from target inventory levels.

The DBRP is NP-hard (Zhang et al., 2017; Ghosh et al., 2017), and being a rich version of this that incorporates more real-life aspects, the DSBRPNI is at least as hard. Thus, finding the optimal decisions, considering the endless possibilities of future scenarios, is complex and in most practical applications, near impossible.

4.2 Example Problem

Figure 4.1 illustrates an example of the problem over a short time horizon with a single vehicle. The figure provides a step by step overview of the system development, with vehicle operations, user demand and roaming. The numbered circles represent stations, where Station 1 and Station 2 are in close enough proximity to be considered neighbors. Users can therefore roam between these two stations. Each station has an inventory capacity of five, represented by the small circles. A filled circle means that there is a bike, while an empty circle illustrates an available lock. The service vehicle has a similar representation of inventory, and the arrow indicates the vehicle's next move. A walking person indicates demand for a bike, while a biker indicates demand for a lock. A time stamp is included in the top right corner.

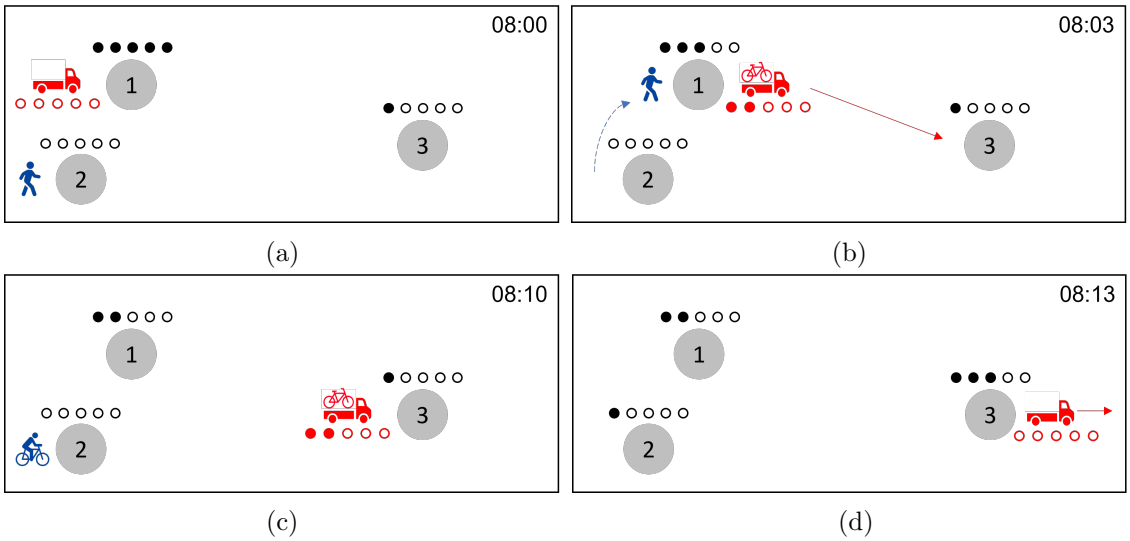


Figure 4.1: Illustration of an example problem with vehicle operations, customer demand and roaming. Adapted from Inngjerdingen & Møller (2022)

In Figure 4.1a the empty service vehicle arrives at Station 1. At 08:00 it begins loading two bicycles from the station. At the same time, a user arrives at Station 2, looking for

a bicycle. Station 2 is empty, but as Station 1 is nearby, the user begins to walk to this station to pick up a bicycle. In Figure 4.1b the vehicle has finished its loading operation. It begins driving to Station 3 at 08:03. It prioritizes Station 3 because this station can not rely on the spillover effect from neighboring stations, as opposed to Station 2. The user, originating from Station 2 picks up a bicycle and leaves Station 1. In Figure 4.1c the vehicle arrives at Station 3 and begins unloading 2 bicycles. At the same time, a user on bicycle arrives at Station 2 and drops of the bicycle here. Finally, in Figure 4.1d, the vehicle is done unloading bikes at Station 3 and is ready for new tasks.

In contrast, if the rebalancing decision did not consider roaming, the service vehicle would likely drive to Station 2 instead of Station 3. This station is located closer to Station 1 and have no bikes available when the decision is made. While it is difficult to say that one solution is better than the other, this option would make the area around stations 1 and 2 well-balanced, while the area around Station 3 would experience a shortage of bikes. With the rebalancing described in Figure 4.1, both of these areas are well-balanced at the end of the time horizon.

Chapter 5

Solution Method

The DSBRPNI is an operational problem that needs to be solved in real time with short computational times. The complexity of this problem makes it difficult to solve, so a common approach is to decompose it into to smaller subproblems. These subproblems are solved sequentially over shorter time horizons using a rolling horizon approach. In this chapter, we introduce a heuristic solution method for the subproblem, applying a novel metaheuristic called *Explorative Preferred Iterative LOokahead Technique* (X-PILOT). The metaheuristic is a modified version of the PILOT framework described in Section 3.4.2, in which deeper branching is conducted to allow for wider exploration of the solution space. In addition, a new integrated approach for multi-vehicle systems is introduced, aiming for more coordinated solutions with better utilization of synergies.

First, the rolling horizon approach and the subproblem are presented in Section 5.1. Second, an overview of the solution method is provided in Section 5.2. Next, sections 5.3 and 5.4 elaborate on how loading decisions and routing decisions are made, respectively. Section 5.5 explains how these decisions are utilized in a construction algorithm as a part of the X-PILOT framework, before presenting a method for evaluating solutions.

5.1 Rolling Horizon and the Subproblem

A common approach to make complex problems more manageable is by consolidating the solutions from a series of subproblems, which are solved using a rolling horizon approach. Each subproblem solves the rebalancing problem over a relatively short time horizon, thereby greatly reducing the complexity. Recall that we solve an online optimization problem, in which new information about demand and station inventory levels are continuously made available during the day. To utilize as much information as possible, decisions should be made at the latest possible time. For the DSBRPNI, this means making loading and routing decisions when arriving at a station. After the loading operations are complete and the vehicle has driven to the next station, new demand has been revealed. Thus, loading and routing decisions can be revisited and solved based on updated information each time a vehicle arrives at a new station. This means that each time a vehicle arrives at a station, we only decide *loading quantities*, and which station to visit next, referred to as the *first-move*. Due to the operational nature of the problem, these decisions should be made within seconds.

The rolling horizon approach is illustrated in Figure 5.1. Each station visit marks a new

decision point, in which a limited time horizon is considered. Upon arrival at the next station, the problem is re-solved with a new time horizon starting from the time of arrival at the station.

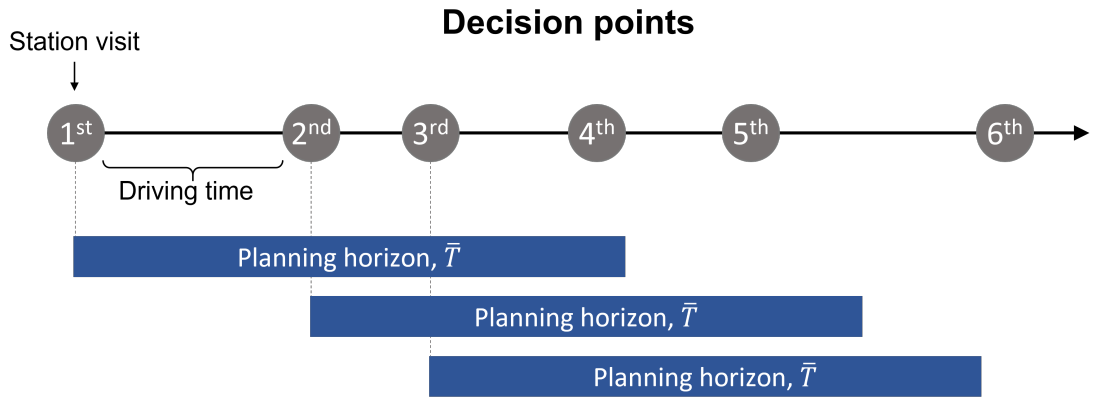


Figure 5.1: Illustration of decision points and planning horizons in the rolling horizon approach.

5.2 Overview of Algorithm

An outline of the solution method is presented in Figure 5.2. After a vehicle arrives a station, the first step is to decide how many bikes to load or unload. After that, a *rebalancing plan* is initialized, along with a tabu list containing the current destinations of all service vehicles. The rebalancing plans are used to look ahead, and create temporary routes and loading operations for all vehicles in the system. Furthermore, rebalancing plans are used to estimate the value of each possible first-move, taking into account future implications.

The next step is to decide a first-move, i.e., which station to visit next. As long as a plan has not reached a defined time horizon or branching depth, the plan is extended by adding new station visits, according to the X-PILOT method described in Section 5.5. First, possible successor stations are identified. Then, a criticality score is calculated for each station, and the most critical stations are selected. New plans are then created by copying the current plan and adding the new stations, and the tabu list is updated for each plan. This process continues in a recursive manner for each plan, until either the branching depth or time horizon is reached. If the time horizon is reached, the plan is added to a list of completed plans. If the maximum branching depth is reached, the plan is completed in a greedy fashion until the end of the time horizon, and then added to the list of completed plans.

Scenarios are now created based on historical data, and all completed plans are evaluated over all scenarios. The evaluation function is described in Section 5.5.2. Based on the evaluation of the plans, a selection criteria is applied and the next station to visit is identified. Note that even though entire plans are evaluated, we are only interested in the first-move.

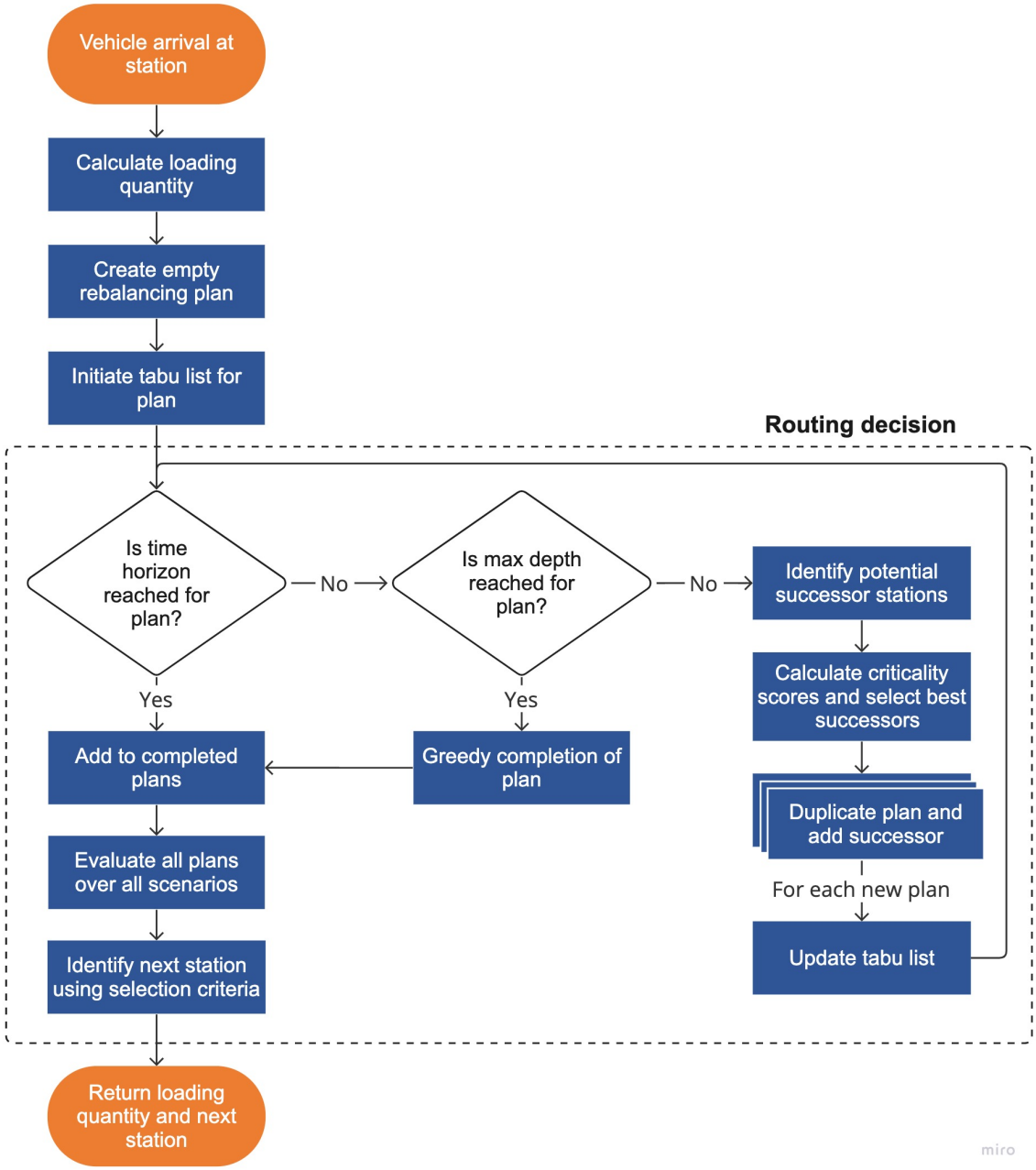


Figure 5.2: Flow chart with an overview of the solution algorithm. The X-PILOT method is applied in the routing decision.

5.3 Loading Decision

The first decision to be made when arriving at a station i is how many bikes to load to (q_{iv}^L) or unload from (q_{iv}^U), the service vehicle v. Upon arrival, the station has an inventory of bikes, L_i^0. In addition, each station has a defined, time specific, target inventory level, L_i^T, which is the optimal number of bikes on the station at the end of the time horizon. Stations with fewer bikes than the target inventory level are defined as delivery stations where bikes are unloaded. Stations with more bikes than the target inventory level are defined as pickup stations and bikes are loaded from these stations. Our approach operates greedily and aims to get the station's inventory as close to the target inventory level as possible.

Algorithm 1: Calculate Loading Quantities

Input: vehicle inventory Q_v^0, vehicle capacity Q_v^V, station inventory L_i^0, station capacity L_i^S, target inventory level L_i^T
Result: loading quantity q_{iv}^L, unloading quantity q_{iv}^U
for *each neighbor in neighborhood* **do**
  if *neighbor inventory* $< 0.1 \cdot$ *neighbor capacity* **then**
    $\delta^{sta} \leftarrow \delta^{sta} + 1$
  else if *neighbor inventory* $> 0.9 \cdot$ *neighbor capacity* **then**
    $\delta^{con} \leftarrow \delta^{con} + 1$
  end
end
if *station inventory* $L_i^0 <$ *target inventory level* L_i^T **then**
  $q_{iv}^U = \min(L_i^T - L_i^0 + \delta^{sta}, Q_v^0, L_i^S - L_i^0,)$
else if *station inventory* $L_i^0 >$ *target inventory level* L_i^T **then**
  $q_{iv}^L = \min(L_i^0 - L_i^T + \delta^{con}, L_i^0, Q_v^V - Q_v^0)$
end

To take user roaming into consideration, the loading and unloading quantities are adjusted according to the number of congested or starved neighboring stations. For a pickup station, the loading quantity is increased in line with the number of congested neighbors, δ^{con}. This means that more vacant locks are made available compared to if roaming had not been taken into account. Neighbors are considered congested if their inventory levels are higher than 90% of station capacity, L_i^S. Likewise, if a delivery station has starved neighbors, the unloading quantity is increased in line with the number of starved neighbors, δ^{sta}. Neighbors are considered starved if their inventory levels are beneath 10% of capacity. Only neighbors that have a similar type of imbalance as the station in focus affects the loading or unloading quantities. Note that δ^{con} and δ^{sta} may be multiplied by an integer number in order to adjust how many extra bikes to pick-up or deliver per imbalanced neighbor.

Furthermore, there are certain logical bounds on loading and unloading quantities. Loading quantities can never exceed the number of bikes at the station, L_i^0. In addition, they are bounded by the number of available slots on the vehicle, i.e., the difference between vehicle capacity, Q_v^V, and the number of bikes already on it, Q_v^0. Unloading quantities can never exceed the vehicle inventory, Q_v^0 and are bounded by the number of available locks at the station, $L_i^S - L_i^0$. Algorithm 1 summarizes the entire approach for calculating loading quantities.

5.4 Routing Decision

After loading or unloading, the next task is deciding which station to visit next. For simplicity, this task can be divided into two main steps. First, a set of potential stations, which is a subset of all the stations in the system, is identified. Second, a criticality score is calculated for each potential station and the stations with the highest scores are returned.

5.4.1 Identifying Potential Stations

Using a tabu list and domain knowledge, stations that are not relevant to visit are filtered out. This is done to reduce the size and complexity of the problem, as we only consider a subset of stations in future calculations. Two empty sets of relevant stations are initialized, one for pickup stations, $\mathcal{N}^P$, and one for delivery stations, $\mathcal{N}^D$. Then, iterating through all the stations in the system, we add stations to $\mathcal{N}^P$ or $\mathcal{N}^D$. Several criteria must be fulfilled in order for a station to be added to one of the sets. First of all, it cannot be in the tabu list. The tabu list is comprised of stations that are either already included on the current route of the vehicle or are scheduled to be visited by other vehicles. Further, a station is defined as a pickup station and placed in $\mathcal{N}^P$ if

$$L_i^0 + D_i > (1 + \rho^S)L_i^T. \tag{5.1}$$

L_i^0 is the actual inventory of bikes at station i and D_i is the net demand for bikes for the next 60 minutes, where a positive net demand indicates demand for locks. ρ^S is a predetermined station cutoff constant and L_i^T is the target inventory for station i. On the other hand, if

$$L_i^0 + D_i < (1 - \rho^S)L_i^T, \tag{5.2}$$

the station is defined as a delivery station and placed in $\mathcal{N}^D$. If a station is self balanced, meaning

$$(1 - \rho^S)L_i^T \leq L_i^0 + D_i \leq (1 + \rho^S)L_i^T, \tag{5.3}$$

the station is omitted from further calculations.

The inventory level at the service vehicle, together with a vehicle cutoff constant, ρ^V, determines which stations we consider as potential stations. If the number of bikes on the vehicle, Q_v^0 is less than $\rho^V Q_v^V$, where Q_v^V is the capacity of the vehicle, the vehicle is nearly empty and potential stations are defined as $\mathcal{N}^P$. This means that the vehicle has to go to a pickup station next. If the number of bikes on the vehicle is greater than $(1 - \rho^V)Q_v^V$, the vehicle is assumed full and potential stations are defined as $\mathcal{N}^D$. Finally, if the number of bikes lies in the interval $[\rho^V Q_v^V, (1 - \rho^V)Q_v^V]$, the vehicle is neither empty nor full and the potential stations are defined as $\mathcal{N}^P \cup \mathcal{N}^D$.

5.4.2 Criticality Score

Once the potential stations have been identified, a criticality score is computed for each station. This score indicates the significance of visiting the station and is composed of five components. The importance of each component can be adjusted based on the decision maker's preferences. The five components for station i are:

- t_i^V: Time to violation
- d_i: Deviation from target inventory level
- n_i: Neighborhood criticality
- D_i: Net demand
- T_{ji}^D: Driving time

Time to Violation

Time to violation, t_i^V, is the first component and is found by calculating the expected time until the next violation occurs at a station. Depending on whether the net demand is positive or negative, the time to violation is calculated in the following way:

$$t_i^V = \begin{cases} \min(\frac{L_i^S - L_i^0}{D_i}, T^*) & \text{if } D_i > 0 \\ \min(\frac{-L_i^0}{D_i}, T^*) & \text{if } D_i < 0 \\ T^* & \text{otherwise.} \end{cases} \tag{5.4}$$

An upper limit of T^* hours is imposed on time to violation in order for normalization of the times to return reasonable values. Normalization of variables is discussed towards the end of this section. However, this means that the model does not differentiate between stations when there is more than T^* hours to violation. Furthermore, time to violation is fixed to T^* hours when net demand is zero. Figure 5.3 illustrates how time to violation is calculated for a pickup station.

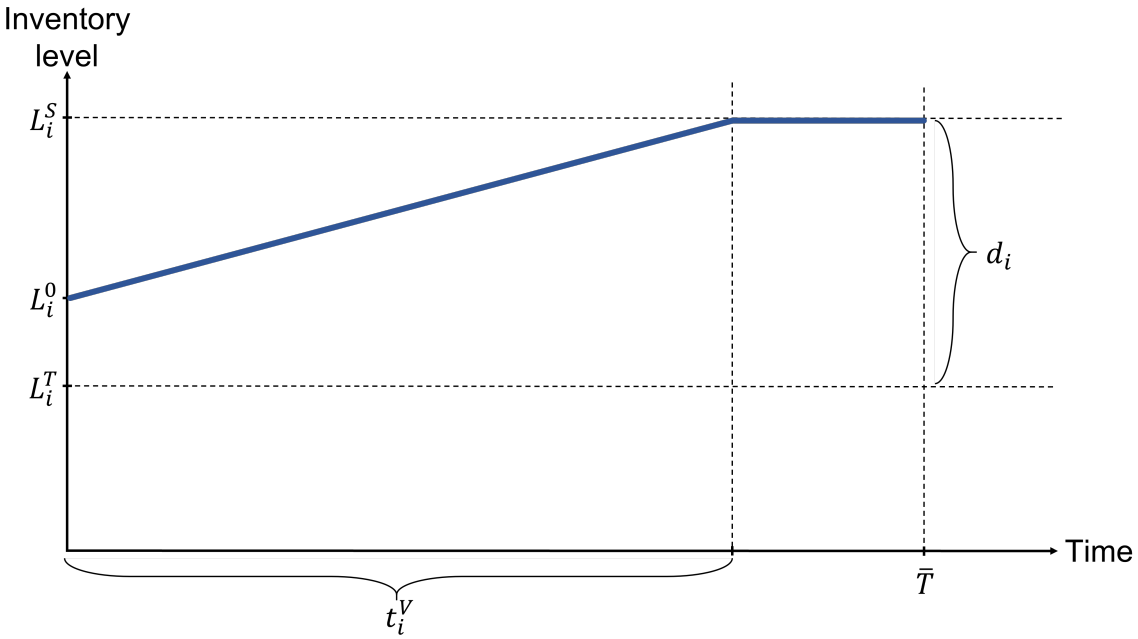


Figure 5.3: Time to violation, t_i^V, and deviation from target inventory level, d_i for pickup station i

Deviation from Target Inventory Level

Deviation from target inventory level, d_i, is the second component of the criticality score. Ensuring that the station's inventory is in line with its target level is crucial for the

model to account for demand beyond its short time horizon, instead of being excessively shortsighted. Historical demand is used to calculate the optimal target level. Note that the calculation of this target level is not in focus in this thesis. Furthermore, the expected net demand, D_i, is included in the calculations to estimate the deviation at the end of the time horizon. Because of this, estimated inventory levels can go beyond station capacity or become negative. To manage this and prevent unreasonably high deviations from occurring, constraints are imposed when stations become full and empty. Equation 5.5 and 5.6 show how the deviation is calculated when the net demand is positive and negative, respectively. In Figure 5.3, the deviation from target inventory level is illustrated for a pickup station.

$$d_i = \begin{cases} |L_i^T - L_i^0 - D_i| & \text{if } L_i^0 + D_i \leq L_i^S \\ L_i^S - L_i^T & \text{otherwise} \end{cases}, D_i \geq 0. \tag{5.5}$$

$$d_i = \begin{cases} |L_i^T - L_i^0 - D_i| & \text{if } L_i^0 + D_i \geq 0 \\ L_i^T & \text{otherwise} \end{cases}, D_i < 0. \tag{5.6}$$

Neighborhood Criticality

The Neighborhood criticality component, n_i, is the third part of the overall criticality score. Rather than considering a single station in isolation, this component looks at an entire area and takes into account interactions between neighboring stations. For each neighbor to the station in focus, a neighbor score is calculated based on four different aspects of neighborhood interactions, before it is added to the neighborhood criticality score. The four components are:

- Neighbor with a similar type of imbalance
- Neighbor which can absorb demand
- Demand at neighbor
- Distance to neighbor

A similarly imbalanced station increases the neighbor score. This means that if the station in focus is a delivery station, the neighbor score increases if the neighbor also is a delivery station. In contrast, a neighbor which can absorb demand reduces the neighbor score. A neighbor can absorb demand in two cases: if it has available locks and the station in focus is a pickup station or if it has available bikes and the station in focus is a delivery station. In addition, if a neighbor station has already been visited in a plan, this reduces the neighbor score. The net demand at the neighbors also affect the neighborhood criticality. For each neighbor, given that the neighbor is the same station type as the station in focus, a demand criticality is calculated using the same logic as in the net demand component, presented in the next section. This score is added to the neighbor score. Finally, all neighbor scores are scaled, based on the distance to the station in focus. Closer stations are more heavily valued than stations further away. The neighborhood interactions are assumed to be stronger when the distances are shorter. This also means that in a cluster with neighboring stations, the station in the center receives the highest criticality score,

when all else is equal, as it has the shortest distance to all the neighbors. The neighbor scores are added together and makes up the neighborhood criticality component.

Figure 5.4 provides an illustration of the components involved in calculating neighborhood criticality for Station 1. With Station 1 being starved, the criticality increases as both Station 2 and Station 3 also exhibit a similar imbalance. Conversely, Station 4 has available bikes and is thereby able to absorb demand, thus reducing the criticality score. Stations 2 and 3 have a negative expected demand, indicating a need for bikes, which further amplifies their contribution to the neighborhood criticality. Since Station 4 does not have a similar imbalance, its demand is not taken into account. Finally, Station 3 is closest to the station in focus, and the criticality contribution from this station is scaled accordingly. Station 4 is furthest away, so the criticality contribution from this station is scaled down compared to the others.

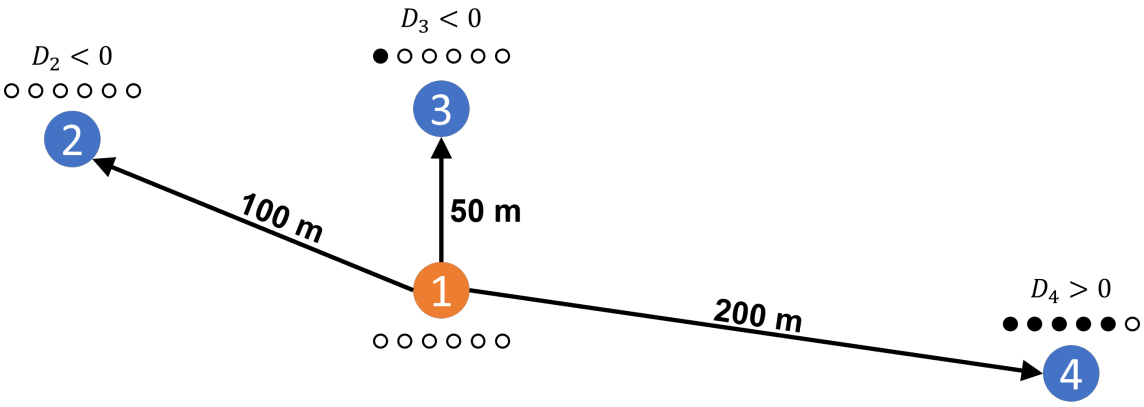


Figure 5.4: Illustration of the components included in the calculation of neighborhood criticality. Station 1 is the station in focus, and stations 2, 3 and 4 are neighbors. Filled and empty dots represent bikes and locks, respectively

Net Demand

In addition to time to violation, deviation from target inventory level and neighborhood criticality, net demand, D_i, is an important measure of the criticality of a station. As an example, consider two potential stations, A and B. Both stations are currently empty and have a target inventory of 12 bikes. However, Station A has an expected net demand of -4 bikes per hour, compared to -1 at Station B. At both stations, time to violation equals 0 and deviation from target inventory is 12 bikes. However, as more violations are expected to occur at Station A in the next hour, it should be prioritized over Station B. For pickup stations, a positive net demand results in a positive contribution to the criticality score, while a negative net demand results in a negative contribution. The opposite is the case for delivery stations.

Driving Time

The driving time, T_{ji}^{D} from the current station, j, to the potential station, i, is the last aspect that is considered. With this component a bias towards shorter driving times can be created. Favoring shorter trips, when all else is equal, can result in more bikes being moved throughout the day and better rebalancing.

After the five components are calculated for all potential stations, they are normalized, meaning their original values are converted to numbers between 0 and 1. For t_i^V and T_{ji}^D, shorter times increase the criticality of a station. However, for the remaining components, higher values increase the criticality. This difference is handled through the normalization process. Finally, the normalized components are multiplied with their respective criticality weights and added together to make up the final criticality scores. The weights of the different components are predetermined and always add up to 1. The final expression of the criticality score is thus a convex combination of the five components, and is presented in Equation 5.7. $\boldsymbol{\omega}^{crit}$ represents the vector of criticality weights, corresponding to the different components.

$$\text{Criticality score} = \boldsymbol{\omega}^{\boldsymbol{crit}} \cdot (t_i^V, d_i, n_i, D_i, T_{ji}^D) \tag{5.7}$$

After criticality scores are calculated for all potential stations, the scores are sorted in descending order. The station with the highest score is chosen and added to the vehicle's route. If multiple routes are created for the vehicle, the algorithm returns a set of stations as per requirement. The stations with the highest scores are then chosen.

5.5 X-PILOT

To avoid making myopic decisions, loading and routing operations are implemented into an adapted PILOT framework, based on the method of Rainer-Harbach et al. (2015) which is presented in Section 3.4.2. The new framework explores more of the solution space around potential routes, resulting in the name *Explorative Preferred Iterative LOokahead Technique* (X-PILOT). Instead of considering single rebalancing moves, *rebalancing plans* consisting of a sequence of moves are created. When a vehicle arrives at a station, several potential first-moves are identified. Each first-move is then extended into rebalancing plans. Next, these plans are evaluated and compared, and the first-move with the best corresponding plans is selected. The selected first-move is the operation which is actually performed by the vehicle. By comparing plans with several subsequent operations, the best sequence of operations over a time horizon is chosen. This means that not only the effects of the very first rebalancing operation is considered, but also how this operation impacts future rebalancing. In this section, a construction algorithm for rebalancing plans is described, before an evaluation function is presented.

5.5.1 Construction Algorithm

An outline of a X-PILOT tree constructed for a single vehicle is presented in Figure 5.5. The first node represents Station 1 at which the vehicle arrives and performs the first loading operation. From here, a *branching* into three possible first-moves is performed, stations 2, 3 and 4. Instead of selecting the best first-move greedily, each first-move is extended by adding two subsequent moves. As branching is performed twice, we define the tree to have a *depth*, α, of two. When the maximum depth is reached, the rest of the plan is constructed greedily until the end of the time horizon, $\overline{T}$. Each path through the tree now represents a rebalancing plan over the given time horizon. There are several different ways of evaluating and selecting the best plan, this is discussed in Section 5.5.2. After

selecting the best branch, the corresponding first-move is selected as the next station visit for the vehicle.

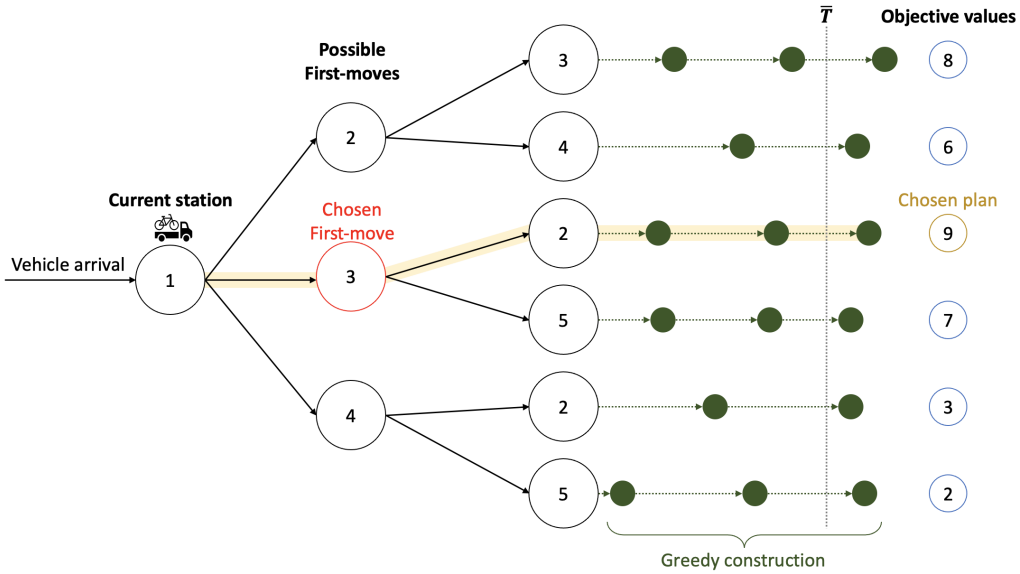


Figure 5.5: Example of a PILOT tree constructed for one vehicle with the X-PILOT solution method. In the example, $\alpha = 2$, $\beta_1 = 3$ and $\beta_2 = 2$. The plan with the highest objective value is chosen. Evaluation and selection of plans is discussed in Section 5.5.2.

Branching

Two key parameters in the X-PILOT construction algorithm are the *branching width* parameter, β, and the *branching depth* parameter, α. The former decides how many successors to be added in each branching and is specific for the depth, while the latter decides how many times branching should be performed. After depth α, the rest of the plans are constructed in a greedy manner until the time horizon is reached. Ensuring that plans are created over an equal length of time is essential for obtaining accurate plan comparisons. This approach allows for a direct assessment of the benefits attained by the plans within the same time frame.

Theoretically, any station could be a possible successor at any time during the construction of the plans. However, a complete enumeration would quickly become impossible due to the computational complexity. Therefore, the width parameter β is used to limit the number of successors. Note that β can take different values for different depths. β_1 is the width at the first depth and determines how many potential first-moves we evaluate. Generally, we would prefer a higher number of successors in the beginning, in order to achieve exploration of many different first-moves. Deeper into the tree, more time has passed and more uncertainty is present. Recall that we are eventually only interested in the first-move, so a wide exploration further into the tree is rather meaningless. The example shown in Figure 5.5 uses $\beta_1 = 3$ and $\beta_2 = 2$. Our general approach is to halve the branching width to the nearest integer after depth one and two. In addition, the width is halved when we consider other vehicles than the one in focus. By making smart selections of successors, as discussed in Section 5.4, we further limit the necessity of high β-values, only looking at the most relevant successors at all times.

To facilitate better diversification in the tree, different criticality weight sets are used in

different main branches. By doing so, a greater variety of solutions can be created, each with different focus areas. For example, some solutions may prioritize time to violation and magnitude of demand, thus focusing on short term gain, while other solutions can take a longer perspective with a main focus on deviation from target inventory levels. Depending on the state of the system and varying demand, the importance of the different rebalancing aspects may vary. By using different sets of criticality weights, the chance of creating solutions that are suitable for any given system state increases.

For each visit in a plan, estimated loading quantities are calculated before the successor stations are chosen. In addition, accumulated customer demand at each station is estimated. This enables a smart selection of successor stations, based on the estimated number of bikes in the vehicle inventory and the estimated number of bikes at each station. Section 5.4 discusses how to select stations based on a known vehicle and station inventory, and the same logic can be applied for the estimated numbers. This also ensures that instead of simply choosing the most critical stations, plans are created taken into account what kind of operations are to be performed. For example, visiting three delivery-stations in a row would limit the number of bikes that can be delivered at each station. Visiting stations in a sequence of e.g., [delivery, pickup, delivery], can often enable a larger number of bikes to be rebalanced per visit. The first delivery creates available space on the vehicle for the upcoming pickup, which again makes bikes available for the following delivery. Note that the estimated loading quantities are only used temporarily to create sensible sequences of visits and to compare the utility gains from the rebalancing plans. The actual loading decisions for all stations are made at time of arrival, i.e., after customer demand has been revealed and more accurate information about the system state is known.

Multiple Vehicles

As discussed in Chapter 3, lack of coordination between service vehicles may lead to suboptimal decision making. Consequently, creating plans for individual vehicles may not provide satisfactory results. Instead, we introduce a method for integrating multiple vehicles into each plan with the aim of exploiting synergies. A rebalancing plan consists of a route for every vehicle in the system, including loading/unloading decisions. The routes are created while taking other vehicles' operations into account. Each plan has a tabu list which is updated for each iteration. The tabu list contains all visited stations for this plan, and the current destinations of all vehicles. This ensures that vehicles do not head to the same stations. An illustration of a rebalancing plan is presented in Figure 5.6.

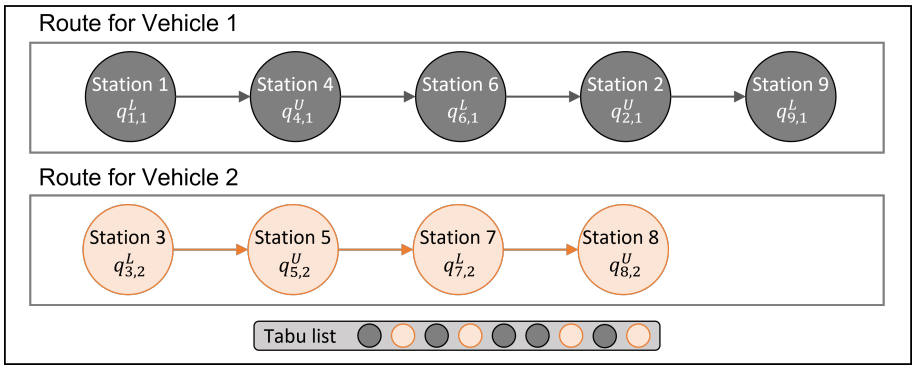


Figure 5.6: A rebalancing plan consists of a route for each service vehicle in the system, including loading decisions. In this plan, loading and unloading is performed every second operation. A shared tabu list is updated throughout the construction of the plan.

The figures 5.7-5.9 illustrate how the X-PILOT method handles a multi-vehicle setting, here illustrated for two vehicles. In Figure 5.7, Vehicle 1 is arriving at Station 1 at time t_0, and we want to decide where it should go next. At the same time, Vehicle 2 is expected to arrive at Station 4 at time t_2.

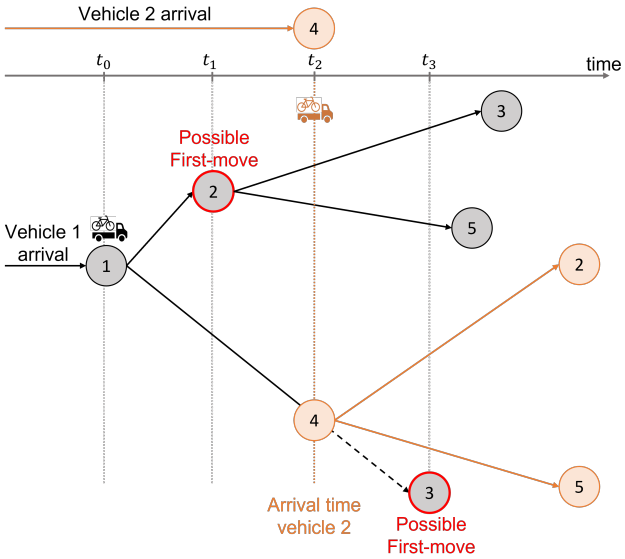


Figure 5.7: First stage of a PILOT tree constructed for 2 vehicles with the X-PILOT approach. Vehicle 1 arrives first, and a decision must be made for its next move. Black nodes and arrows denote station visits and routes for Vehicle 1, while operations of Vehicle 2 are illustrated by orange color. Dashed lines represent pruned branches.

- First, we identify two potential first-moves for Vehicle 1; Station 2 and Station 3. Depending on the driving distance, expected arrival times differ for these stations, which is indicated by their placement on the time-axis. The idea now is to identify the next upcoming event for each branch.
- The top branch involves a rather short trip, and Vehicle 1 is expected to arrive at Station 2 at time t_1, before the arrival time of Vehicle 2. Therefore, this branch is extended in a normal fashion and Station 3 and Station 5 are added.
- The lower branch however, involves a longer trip, meaning that Vehicle 2 is expected to arrive at Station 4 before Vehicle 1 arrives at Station 3. This means that the routing decision of Vehicle 2 must be made before the next routing decision of Vehicle 1. Therefore, we add the vehicle arrival of Vehicle 2 to the branch at time t_2. We now perform a branching for Vehicle 2, while keeping in mind that Vehicle 1 is on route to Station 3. Two possible successors are selected for Vehicle 2; Station 2 and Station 5. Due to the arrival of Vehicle 2, the original black branch is pruned (illustrated by a dashed line), and we continue the construction of the tree from the orange branches of Vehicle 2. Even though the branch is pruned, the station visit is still included in the plan.

In Figure 5.8 we continue building the solution tree with the same logic as before.

- On the top two branches, Vehicle 2 is expected to arrive at Station 4, before Vehicle 1 arrives Station 3 or Station 5. Again, we need to make routing decisions for Vehicle 2 before we continue with Vehicle 1. The arrival of Vehicle 2 is therefore added to the branches originating from Station 2, and branching for Vehicle 2 is performed from there. We prune the black branches, but make note of the destinations of Vehicle 1.

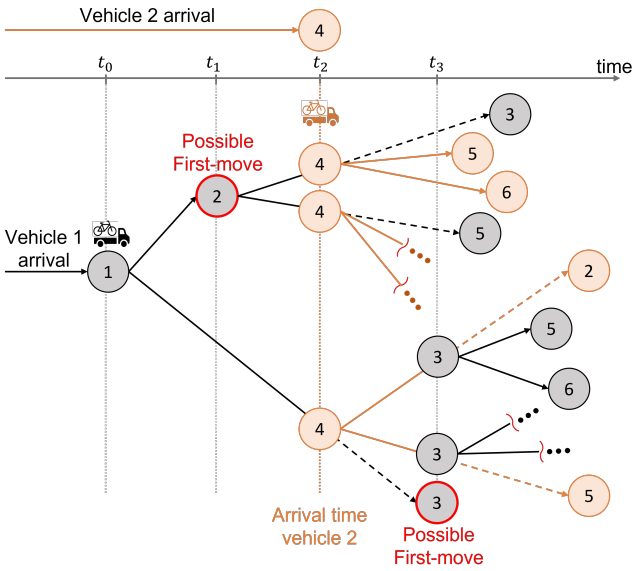


Figure 5.8: Second stage of tree construction. Branching occurs for the vehicle that is expected to arrive its designated station first. Some branches have been cut off to simplify the figure, illustrated by three dots.

To simplify the figure, some branches have been cut of. This is illustrated by three dots.

- In the lower two branches, the visit at Station 3 reappears as Vehicle 1 is expected to arrive at this station before Vehicle 2 is expected to arrive Station 2 or 5. The orange branches are pruned and branching continues from Station 3.

After the given depth is reached and branching is complete, the rest of the routes are created greedily until the end of the time horizon, $\overline{T}$. This is illustrated with green nodes and dotted lines in Figure 5.9.

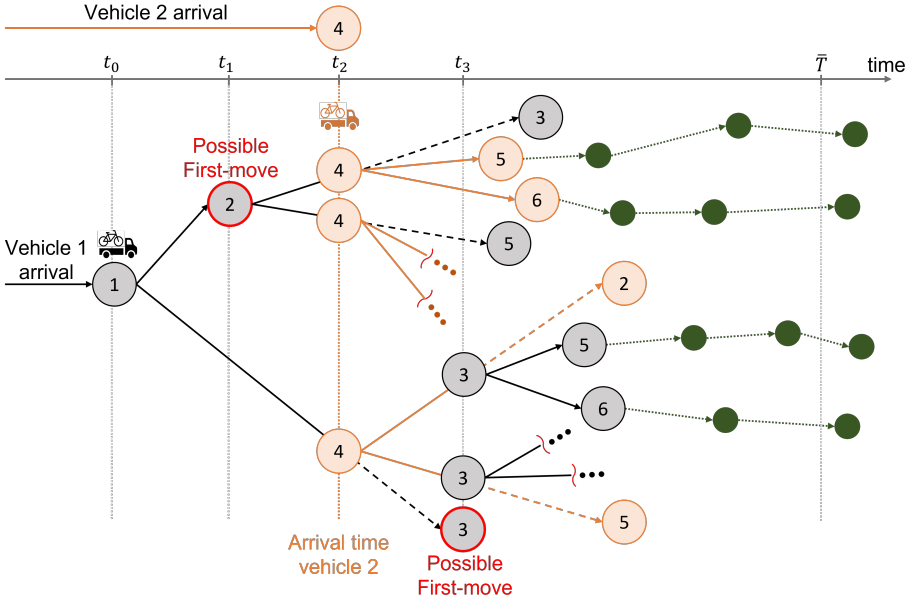


Figure 5.9: Final stage of tree construction. The rest of the routes are created greedily until the end of the time horizon $\overline{T}$, illustrated by green nodes and dotted lines.

Each path through the PILOT tree corresponds to a plan, which includes routes for all

the vehicles in the system. The concept can easily be extended for more vehicles, using the same logic.

5.5.2 Evaluation Function

The construction algorithm creates a set $\mathcal{P}$ of different plans for rebalancing. Each plan, p, consists of a set of routes $\mathcal{R}_p$ for different vehicles, and each route is made up of a set of station visits K_{pr}. Ideally, we would like to return the first-move with the best corresponding plans. To identify these, the evaluation function evaluates the quality of each plan, $p \in \mathcal{P}$ by calculating the utility gain from the vehicles' station visits and rebalancing operations. Plans are evaluated over multiple demand scenarios, $s \in \mathcal{S}$, and finally the best plan is determined by a specific selection criterion.

Stochasticity and Scenarios

As discussed in Chapter 4, uncertainty in demand makes the rebalancing problem stochastic. Rather than simply using expected demand in the evaluation function, we therefore introduce scenarios. A scenario is a realization of customer demand for bikes and locks and is generated using sampling. For each station in the system, we know the expected arrival and leave intensity for bicycles, for the considered time horizon. These rates are based on historical data. With these expected values, we create realizations of net demand, by sampling arrivals and departures from Poisson distributions. One scenario is comprised of one realization of net demand, during the time horizon, for every station in the system. Changes in demand naturally impact the effect of rebalancing operations, so each plan is evaluated for all scenarios to see how the plan performs for different realizations of demand. When a plan is evaluated for a given scenario, calculations are made as if demand is deterministic. Using scenarios enables a more realistic and flexible solution method that can capture the variability of the BSS. This can lead to robust rebalancing plans, where the corresponding decisions perform well across a wide range of scenarios, rather than being optimized for a single realization of demand.

Evaluation of Rebalancing Plans

After both the rebalancing plans and demand scenarios are generated, the evaluation is performed. The entire procedure can be summarized by Algorithm 2.

Essentially, the function estimates the difference between the utility that would occur with and without the rebalancing operations for each demand scenario. This difference in utility is for each plan and scenario denoted U_{ps}. The evaluation considers three aspects of utility from each station visit $k \in \mathcal{K}_{pr}$ in route r of plan p. This is *avoided violations*, Δv_k, *enabled roaming*, Δr_k, and *reduced deviation* from target inventory levels, Δd_k. Note that only stations which are impacted by rebalancing are considered, as all other stations are assumed to be indifferent. This means that calculations are only made for a limited number of stations, restricting computational complexity.

Avoided violations imply that there are violations that are expected to occur within the time horizon, but that can be avoided by the rebalancing operations in a plan. This is illustrated by the curves in Figure 5.10. The blue curve shows the development of station inventory level given that no rebalancing is performed. The station experiences demand

Algorithm 2: Evaluation Function

Input: plans $p \in \mathcal{P}$, scenarios $s \in \mathcal{S}$
Result: plan utility, U_{ps}
for *each plan* $p \in P$ **do**
  for *each scenario* $s \in \mathcal{S}$ **do**
    for *each route* $r \in \mathcal{R}_p$ **do**
      for *visit* $k \in \mathcal{K}_{pr}$ **do**
        $\Delta v_k \leftarrow$ violations without visit - violations with visit
        $\Delta d_k \leftarrow$ deviation without visit - deviation with visit
        for *each neighboring station* $n \in \mathcal{N}$ **do**
          $\Delta r_{kn} \leftarrow$ roaming without visit - roaming with visit
          Scale Δr_{kn} depending on distance to neighbor, n
        end
        $\Delta r_k \leftarrow \sum_{n \in \mathcal{N}} \Delta r_{kn}$
      end
      $U_{sr} \leftarrow \sum_{k \in \mathcal{K}_r} \gamma_k (\omega^{\mathrm{v}} \Delta v_k + \omega^{\mathrm{r}} \Delta r_k + \omega^{\mathrm{d}} \Delta d_k)$
    end
    $U_{ps} \leftarrow \sum_{r \in \mathcal{R}_p} U_{sr}$
  end
end

for bikes, and the inventory is empty by time t_2. All demand for bikes occurring after t_2, indicated by the dashed line, leads to violations. The orange curve, however, shows station inventory level when a rebalancing operation is performed at time t_1. Bikes are delivered to the station, so that the station inventory is increased. Now, there are no violations occurring within the given time period. In other words, the violations are avoided. Note that only violations occurring *after* the station visit can be avoided. If the station was visited some time after t_2, the violations that had already happened could naturally not have been avoided.

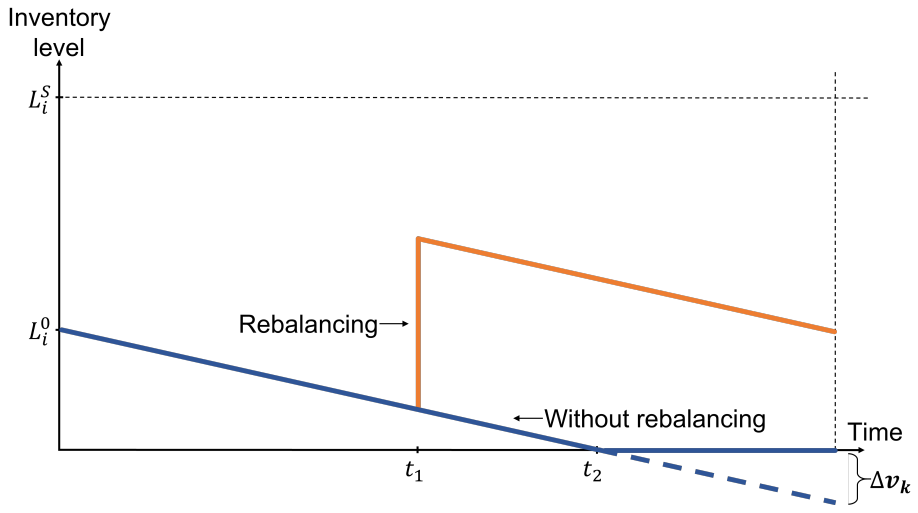


Figure 5.10: Station inventory curve with and without rebalancing. With rebalancing, Δv_k violations can be avoided.

Even if a station is not visited during rebalancing, it may benefit from the rebalancing of its neighbors through roaming. Consider an area with several empty stations. If bikes are made available for at least one station in this area, then users from the neighboring

stations may roam to the rebalanced station to pick up a bike. Without rebalancing, all demand would lead to lost trips due to starvation. However, after rebalancing, demand at the other stations can be handled through roaming. The benefit from the roaming is scaled by distance, so that a short roaming implies a comparatively larger benefit than a long distance roaming. Roaming can occur not only in a starved area where bikes have been delivered, but also in a congested area where locks have been made available. Note that roaming is only possible *after* the neighboring station has been rebalanced. Demand happening before the time of the rebalancing operation cannot be handled by roaming.

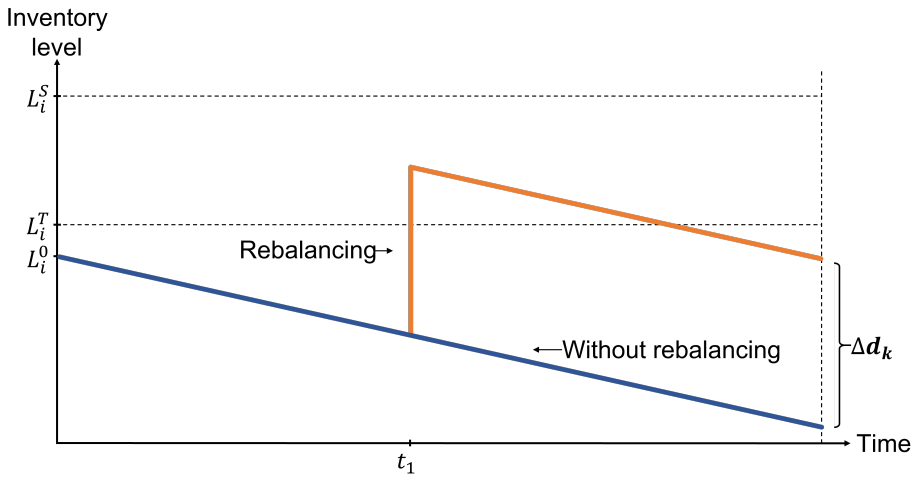


Figure 5.11: The curves illustrate station inventory levels with and without rebalancing at time t_1. Deviation from target inventory level is reduced by Δd_k due to the rebalancing operation.

The last element of the evaluation function is related to target inventory levels. If a rebalancing operation takes bike inventory at a station closer to the target inventory level, then this is considered a benefit. The closer the station gets to its target inventory level, the greater the utility gain. This concept is illustrated in Figure 5.11. The deviation from target inventory level, L_i^T, at the end of the time period is relatively large if no rebalancing is performed. However, a smaller deviation can be obtained by a rebalancing operation. The gain in utility is equal to the reduction in deviation, Δd_k.

When a plan is evaluated, all station visits in all vehicle routes are considered. However, these visits naturally occur at different times. The first visits in a route are considered most important, as these are close in time and hence subject to limited uncertainty. As the future becomes more distant, uncertainty related to station inventory and user demand increases. Therefore, a discounting factor, γ_k, is implemented in the evaluation function. The formula for calculating γ_k is presented in Equation 5.8, where i denote the discount rate.

$$\gamma_k = \frac{1}{(1+i)^k} \tag{5.8}$$

There is no general answer as to what an optimal solution is. Opinions on the relative importance of violations, roaming and target inventory levels may differ between decision makers. Weights for each factor, ω^v, ω^r and ω^d, are introduced in the evaluation function so that the relative importance between them can be altered.

Selection Criteria

After all plans have been evaluated for all scenarios, the task of deciding the best first-move remains. This decision process is an important component of scenario based approaches according to Pillac et al. (2013). Recall that the first-move is the first trip for the vehicle that has just arrived at a station. Normally, each first-move is part of several different plans. Selecting the best first-move is not a trivial task, and different selection criteria can be applied. Some possible approaches include:

1. Evaluate each plan over all scenarios. Identify the plan that performs best on average, and select the corresponding first-move. Inspired by the definitions of Pillac et al. (2013), we call this approach *Expectation*.
2. Evaluate each plan over all scenarios. Find the best plan and corresponding first-move for each scenario. Select the first-move that is returned most often, i.e., the first-move that is best in most scenarios. This approach is defined as *Consensus*.
3. Evaluate each plan over all scenarios. For each first-move, save the lowest value across all scenarios. Select the first-move that has the highest minimum value. This is the *Maximin* approach.

There are various advantages and disadvantages to consider for the different approaches. *Expectation* (alternative 1), use averages that can provide consistent results of high quality. However, averages may be influenced by outliers, meaning that a good first-move can be excluded if it performs badly in only a few scenarios. By selecting the solution that performs best in most scenarios, such as *Consensus* (alternative 2), there should be a high likelihood of achieving high quality rebalancing moves that are well suited for a broad range of demand realizations. The *Maximin* approach (alternative 3), is likely robust, but may not provide solutions of the highest quality. Keeping in mind that bicycle rebalancing is an operational task that is performed many times every day, being risk averse makes little sense. Therefore, only *Expectation* and *Consensus* are utilized in this thesis.

Chapter 6

Simulation Framework

When solving the subproblem, the solution provides decisions on loading quantities and which station to visit next. However, assessing the quality of this solution in isolation is challenging. To evaluate the long term solution quality and the performance of our solution method in a realistic setting, we therefore employ a simulation framework that utilizes discrete-event simulation (Bakker et al., 2022) and imitates real-world BSSs. This allows us to test and tune our solution method as a rebalancing policy and compare it to other policies. First, in Section 6.1, a general overview of the simulator is given. Section 6.2 describes a module that is added to the simulator in order for it to handle roaming. Finally, in Section 6.3, we propose a method to evaluate policies used in the simulator.

6.1 Overview of Simulator

By keeping track of bike trips and rebalancing actions the simulator imitates a real-world BSS with stations, bicycles, users and service vehicles. Each step of the simulation process is triggered by a discrete event. There are three types of events in the simulator, *bike departures*, *bike arrivals* and *vehicle arrivals*. First, the simulator draws a customer arrival scenario for each station using a Poisson process with a rate based on historical demand. A scenario consists of a list of bike departure events, which includes times of departure. All departure events are added to an event queue, which is sorted based on the respective departure times. A vehicle arrival event at time zero is also added to the queue for each vehicle. The simulator works by pulling events from the queue in chronological order and performing a set of actions depending on the event and the state of the system. Figure 6.1 visualizes this iterative process.

If a vehicle arrival event is pulled, the rebalancing policy, which solves the DSBRPNI subproblem, is called. The policy solves the problem using snapshot data from the simulator as parameters as well as historical data. The output from the policy includes how many bikes to pick up or drop off at the current station, and which station to visit next. This information is taken into the simulator, which performs vehicle operations accordingly. If a bike departure is pulled from the queue and accepted, a bike is removed from the station and a bike arrival event is created. These two events together constitute one bike trip. The arrival station is drawn from a stochastic distribution, based on the probability of a trip between the given stations. These move probabilities between stations are based on historical customer trips. The duration of the trip is also calculated and the bike arrival event is added to the queue. Finally, when a bike arrival is pulled and accepted, the bike

is parked and the trip is finished.

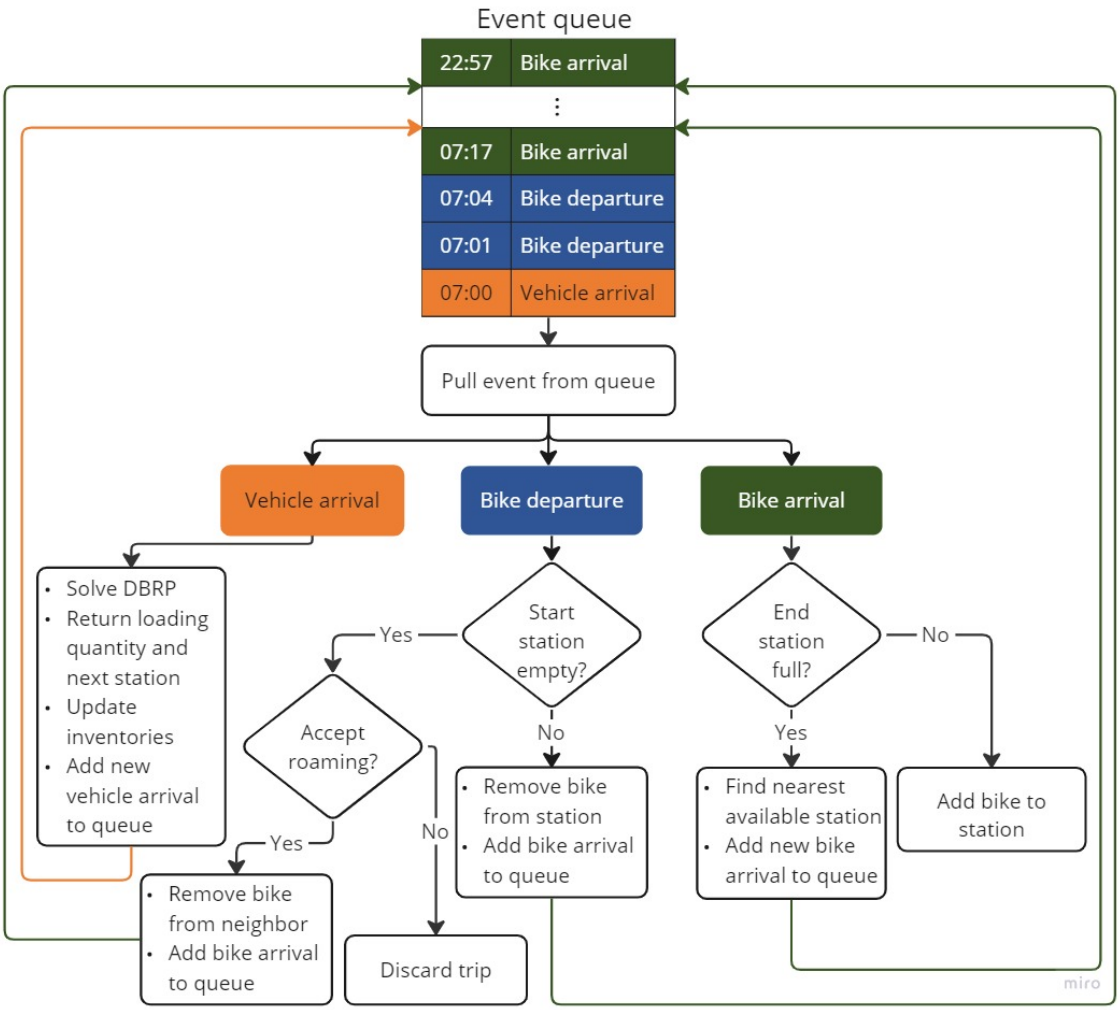


Figure 6.1: Overview of how the simulator pulls events from event queue and performs actions

6.2 Roaming Module

The spillover effect is a central part of this thesis and the solution method presented in Chapter 5. However, previous models have commonly ignored this effect and only looked at starvations and congestions at stations. In the case of a congestion, the bike has disappeared from the system, instead of being relocated to another station. In the simulator however, a more correct logic is incorporated. Here, a new bike arrival event is created at the nearest station with an available lock. This is important as bikes cannot simply vanish from the system, when users arrive at congested stations. Since the user *has* to deposit the bike at a station, the roaming distance can be high if all the nearby stations are full.

In the case of a starvation, the potential customer has been assumed to leave the system in previous models. Since we consider the possibility for a user to roam to a neighboring station if the preferred station is starved, and since we consider this to be less severe than abandoning the system, it is important to incorporate this effect into the simulator. Only

then can we accurately measure the quality of our solutions and the value of considering neighborhood interactions. Therefore, a roaming module is added to the simulator. With a probability depending on the distance, this module sends users to the nearest neighboring station with bikes in case of a starvation. A function, $p(x)$, describing the relationship between the distance to the nearest station, x, and the probability of acceptance, is used in an acceptance-rejection method. In Figure 6.2, this method is illustrated. If a customer arrives at a starved station, the distance x to the nearest neighbor with bicycles is calculated. Then a random number is drawn from a uniform distribution between 0 and 1. If this number is less than or equal to the probability of acceptance for distance x, the user walks to this neighbor. If not, the user leaves the system and a starvation is recorded.

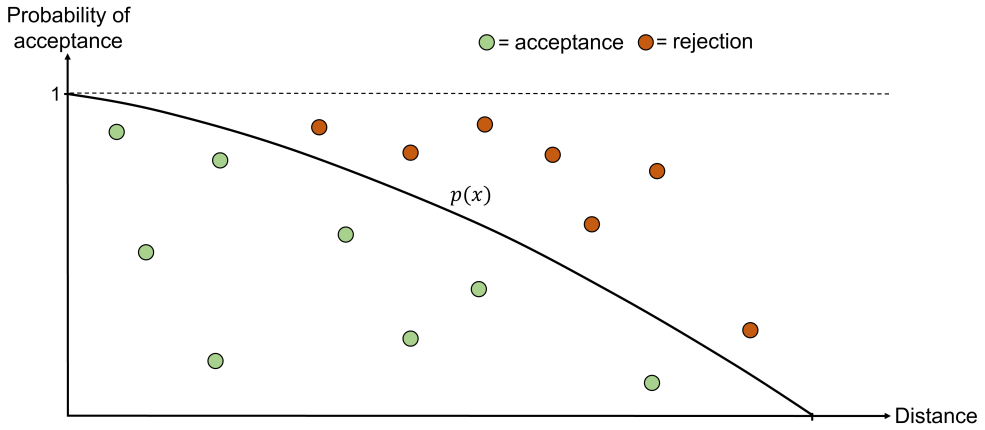


Figure 6.2: The acceptance-rejection method used in the simulator for determining whether a roaming event occurs when a station is starved

6.3 Evaluation of Policies

To evaluate a rebalancing policy, the simulator keeps track of the outcomes of all bicycle arrival and departure events during a simulation. Most importantly, the simulator records all starvations that occur, as well as roaming for bikes and locks. The distances of the roaming actions are also recorded. After running the simulator over a predefined length, the magnitude of starvations and roaming can be evaluated. Comparing results from different parameter configurations gives a good foundation for tuning of parameters. In addition, our algorithm can be compared to other benchmark policies which use different solution methods. Based on these comparisons, the relative performance of our model can be evaluated. Section 8.1 explains in detail which evaluation metric that is used and how these comparisons are done.

Chapter 7

Case Study

To test the developed solution method, numerical values are needed for all parameters. In addition, sets containing vehicles and stations must be initialized according to the specific instance. The process of determining the various input parameters is presented in Section 7.1. Further, Section 7.2 presents key data on the selected test instances, while Section 7.3 describes an example solution to a subproblem for one of these instances.

7.1 Input Data and Parameters

Several types of input parameters and data influence the model. These are related to infrastructure, such as service vehicles and stations, as well as user demand, initial states, target inventory levels and weights for the criticality and evaluation functions. In this section, the values for these parameters are discussed. Real-world data and parameters from Urban Sharing are utilized when applicable.

7.1.1 Stations and Driving Time

Information regarding geographical locations and station capacities are collected from Urban Sharing's databases for the Trondheim, Bergen and Oslo instances. Similar information for the New York instance is gathered from open source data. Travel times between stations are calculated based on the distance between stations and a given speed. Most of the rebalancing operations are performed in city centers where traffic runs relatively slow, so the service vehicle speed is set to 15 km/h and a parking time of 1 minute is added to the driving time. The speed of bikes is set to 7 km/h and walking speed is 4 km/h.

7.1.2 Roaming and Neighboring Stations

As discussed in Section 6.2, it is assumed that the willingness of users to walk in order to find an available bike is dependent on the walking distance. Costa Affonso et al. (2021) conducted a survey on how far people are willing to walk from an empty station in order to find a bike. The results are shown in Figure 7.1. As seen in the figure, there are few people who are willing to walk further than 500 metres, and most are not willing to walk more than approximately 350 metres. Based on these data, a distance limit of 350 metres is set for which stations that are considered neighbors during rebalancing calculations.

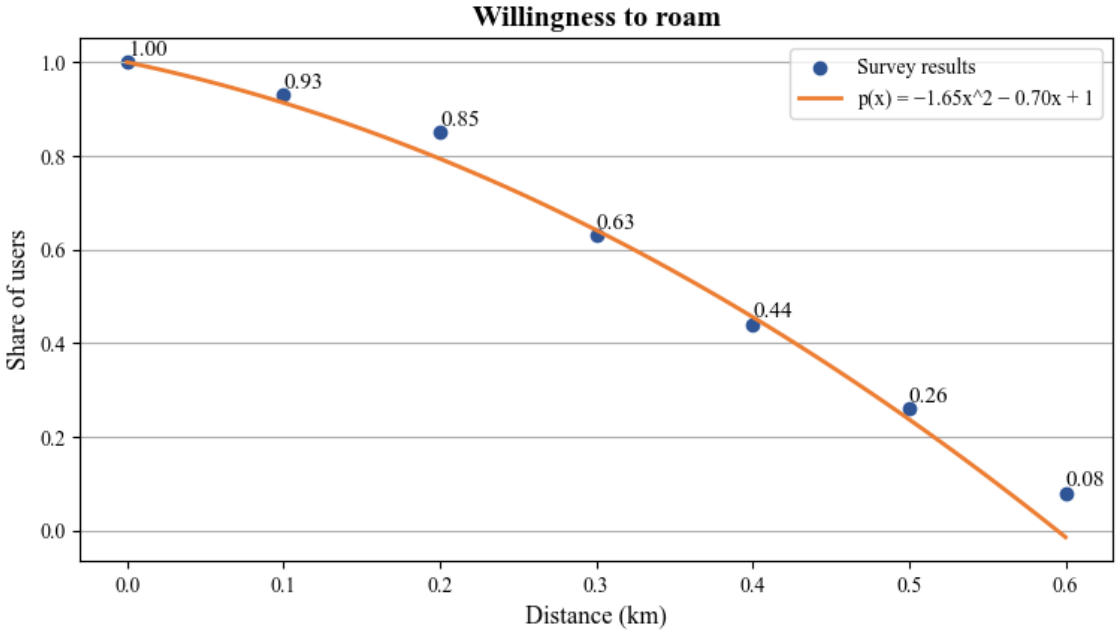


Figure 7.1: Share of users roaming to neighboring stations in search of a bike dependent on walking distance

In the simulator, however, whether a customer chooses to roam or not is subject to uncertainty and not determined by a fixed distance limit. Section 6.2 explains the acceptance-rejection method used to generate these roaming events. Based on the survey results from Costa Affonso et al. (2021), a quadratic function, $p(x)$, is constructed using regression. The following function is used, which returns the probability of a roaming event when x is the distance to the neighbor in kilometres:

$$p(x) = -1.65x^2 - 0.70x + 1. \tag{7.1}$$

7.1.3 Service Vehicles and Handling Time

The solution method can be used with both a single service vehicle and in a multi-vehicle system. Both homogeneous and heterogeneous fleets are supported, but the study is conducted with identical service vehicles with a capacity of 20 bikes, as this is used in Urban Sharing's BSSs. Gleditsch et al. (2022) study the handling time for loading and unloading bikes, and conclude that this varies depending on individual workers and the proximity of parking. Furthermore, it is not necessarily linear with number of bikes handled. However, in order to simplify the model they assume a linear relationship between handling time and number of bikes handled. The same strategy is utilized here, with a unit handling time set to 0.5 minutes.

7.1.4 User Demand

User demand data is based on real-life historical data provided by Urban Sharing and open source data. The data sets are based on historical hourly demand for Oslo, Trondheim, Bergen and New York for weeks 31, 34, 35 and 31 respectively. Demand data is given as arrive and leave intensities of bicycles per hour for each station. Customer events are generated with these intensities used as the mean number of events in a Poisson

distribution. The scenario generation described in Section 5.5.2 is also based on these data.

Each station i in the BSS has a probability distribution that specifies the likelihood of a trip initiated from that station ending at any other station j. This probability is denoted as P_{ij}. Since every trip must have an end station, the sum of probabilities for all possible end stations j must equal one for each station i in the set of all stations, $\mathcal{N}$. This is shown in Equation 7.2. These probabilities are also provided by Urban Sharing, and based on real-life data.

$$\sum_{j \in \mathcal{N}} P_{ij} = 1 \qquad i \in \mathcal{N} \tag{7.2}$$

7.1.5 Initial State

The initial load of bicycles at the stations and service vehicles are updated each time a vehicle arrives at a station, i.e., before each run of the rebalancing model. It is also necessary to know the location of each vehicle, and the time left until arrival at the next station. In practice, these data are set to the updated real-world values. When the problem is simulated and solved in a rolling horizon fashion, initial state is given by the current state of the simulator. The upcoming station visit for a service vehicle is locked, and the remaining time until arrival is calculated based on the completed driving time from the previous station visit.

7.1.6 Target Inventory Level

There are several possible approaches for setting a target inventory level for a station in a given time horizon. As this is not the focus of this thesis, a method from Urban Sharing is used to calculate the target inventory levels. This method sets the target inventory level of a station in a time period in such a way that the probability of a starvation equals the probability of a congestion. The formulation is provided in Equation 7.3.

$$L_{it}^T = \begin{cases} \frac{\sigma_{it}^B (L_i^S - \mu_{it}^L) + \sigma_{it}^L \mu_{it}^B}{\sigma_{it}^B + \sigma_{it}^L}, & \text{when } \sigma_{it}^B, \sigma_{it}^L > 0, \\ L_i^S/2, & \text{otherwise,} \end{cases} \quad i \in \mathcal{N}, t \in \mathcal{T} \tag{7.3}$$

Here, μ_{it}^B, σ_{it}^B, μ_{it}^L and σ_{it}^L are the estimated demand and standard deviation for bikes and locks, respectively. L_i^S is the capacity at station i.

7.1.7 Length of Time Horizon

When solving the subproblem, the length of the considered time horizon is a modeling choice. If a short time horizon is used, only user demand occurring in the near future is considered. With a longer time horizon, the model looks further ahead. In Chapter 8, subproblems with various time horizons, from 10 to 60 minutes, are studied. A discounting factor is also applied in order to prioritize events that are closer in time, as described in Section 5.5.2. Different discount rates are also tested in the computational study.

7.1.8 Weights for Criticality Score and Evaluation Function

There are five weights in the calculation of the criticality score presented in Section 5.4.2. The values of these weights impact which stations that are prioritized for rebalancing operations. Deciding the values of these weights is not a trivial task, but they have a significant impact on the quality of the rebalancing. Therefore, it makes sense to perform a study where different values are analyzed through simulation. This study is performed in Section 8.2.3.

Similarly, three weights are used in the function that evaluates solutions, as described in Section 5.5.2. Different weights can lead to different rebalancing operations being chosen, and thereby directly impact the solution quality. The choice of weights is to some degree dependent on the preferences of decision makers, and how they rate the importance of different types of violations. A study on different weight sets for the evaluation function is performed in Section 8.2.1.

7.2 Test Instances

Real data from the BSSs in Trondheim, Bergen and Oslo provided by Urban Sharing, as well as open source data from Citi Bike NYC, are used for the computational study in Chapter 8. Table 7.1 describes the main characteristics of these instances, and the city maps are shown in Figure 7.2.

Table 7.1: Characteristics of test instances

City	# stations	# bikes	Week
Trondheim	66	768	34
Bergen	100	1,000	35
Oslo	253	2,885	31
New York	931	14,421	31

The BSS in Trondheim is the smallest instance, and is a relatively well balanced system by itself. If the system is run for five days without rebalancing operations, there is an average rate of 95.1% successful events, according to our simulations. A definition of successful events is provided in Section 8.1. The system's ability to self-balance is supported by the fact that most of its stations are situated at roughly the same elevation. However, some stations are outliers located far from the city center and at a significantly higher altitude above sea level. These stations tend to experience starvation if no rebalancing is performed.

Bergen BSS is somewhat larger than Trondheim BSS, with 100 stations and 1,000 bikes. Most stations are located in the city center, with rather short distances between them. This means that each station in the city center has many neighbors. When Bergen is simulated for five days without rebalancing operations, there is a rate of 97.2% successful events. This means that also Bergen BSS has a high degree of self-balance.

Compared to Trondheim and Bergen, the BSS in Oslo is considerably larger, with over 250 stations and close to 2,900 bikes. Travel patterns in Oslo make the system far less self-balanced than in the other cities. The city center is situated at sea level, and contains a lot of businesses. This leads to a high rate of incoming bikes, especially during the morning rush hours. In contrast, the areas outside of the city center consist mainly of

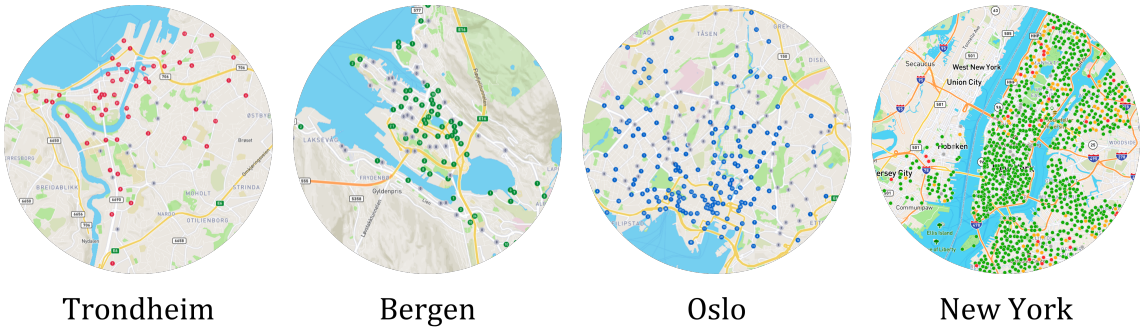


Figure 7.2: Station maps for BSS's in Trondheim, Bergen, Oslo and New York

residential areas that are located at higher elevations. Since users are more inclined to cycle downhill, these areas tend to suffer from a shortage of bikes. As a result, five days of simulation in Oslo without rebalancing leads to only 75.5% successful events.

Finally, Citi Bike NYC is the largest BSS in the United States, and by far the largest instance utilized in the computational study. The system consists of 953 stations and more than 14,000 bikes which are spread across a wide geographical area including Manhattan, Brooklyn, Queens, the Bronx, Jersey City, and Hoboken (Citi Bike NYC, 2023). Five days of simulation on this instance, without rebalancing, results in a service rate of 94.9% successful events.

Each of the different BSS's can be used with different configurations. Notation for test instances is therefore introduced in Table 7.2. The notation includes city abbreviation, week number and the number of service vehicles. For example, `TD_W34_1V`, indicates Trondheim BSS in week 34 with one service vehicle.

Table 7.2: Examples of notation used for test instances

Instance	City	Week	# vehicles
`TD_W34_1V`	Trondheim	34	1
`BG_W35_1V`	Bergen	35	1
`OS_W31_2V`	Oslo	31	2
`NY_W31_5V`	New York	31	5

7.3 Example Solution to a Selected Test Instance

Figure 7.3 illustrates a rebalancing plan created as a solution to a given subproblem for the test instance `TD_W34_1V`. The time horizon is 25 minutes and the empty service vehicle has just arrived Station 35 when the subproblem is solved. The solution can be summarized with the following actions:

- Load 4 bicycles from Station 35
- Drive to Station 54
- Load 8 bicycles from Station 54
- Drive to Station 40
- Unload 3 bicycles onto Station 40
- Drive to Station 27

- Unload 9 bicycles onto Station 27

Recall that we are only interested in the initial loading quantity and the first-move, so the decisions made from this plan would be to load four bicycles at Station 35, and then drive to Station 54. Later, upon arrival at Station 54, a new, similar, subproblem is solved, but now with updated information about the state of the system.

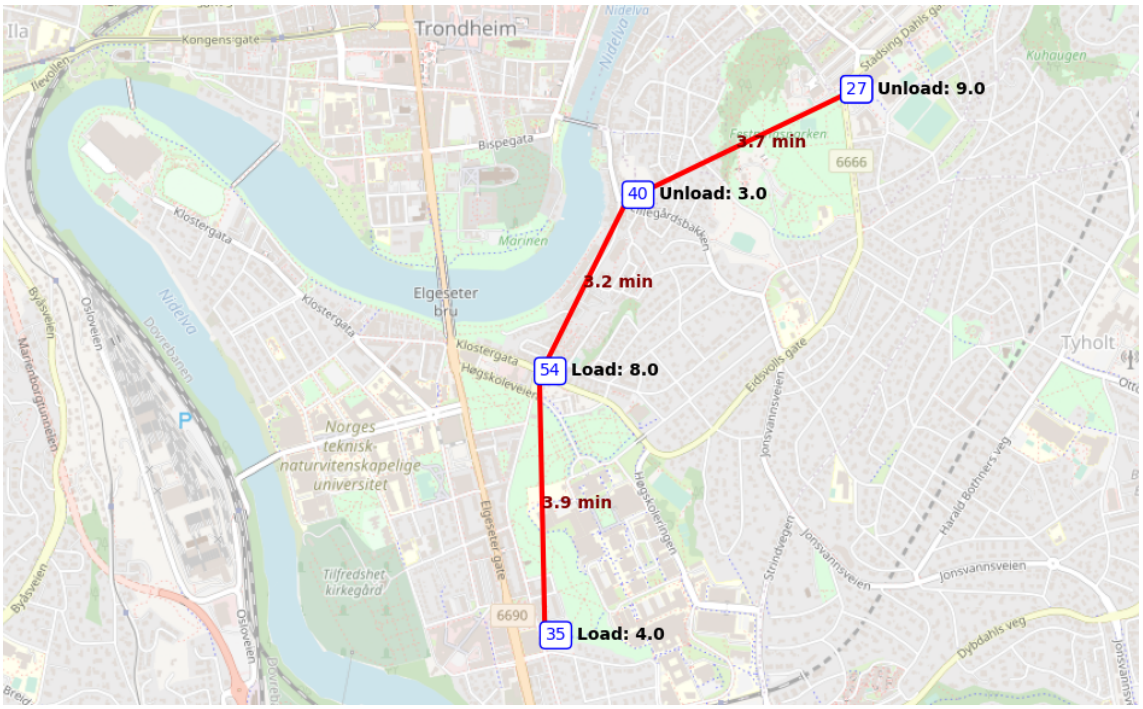


Figure 7.3: Example solution for a given subproblem

Chapter 8

Computational Study

In this chapter, a computational study is performed on the solution method described in Chapter 5. First, in Section 8.1, the evaluation metrics used for the computational study are presented. In Section 8.2, input parameters are tested in order to identify values for evaluation function weights, discounting factors and criticality weights that perform well over all the studied instances. Then, an analysis is performed in Section 8.3 to study how the different X-PILOT modelling parameters affect the solutions in terms of solution quality and computational time. Finally, the computational performance is more thoroughly examined in Section 8.4 using larger test instances and more service vehicles.

The solution method is implemented in Python using Visual Studio Code IDE. Specifications of the computer hardware and software are presented in Table 8.1.

Table 8.1: Specifications of hardware and software used for computational study

CPU	2x 2.4GHz Intel Xeon Gold 5115 CPU – 10 core
RAM	96GB
Operating System	CentOS 7.9.2009
Python version	Python 3.9.6

The complete code is available on Github: https://github.com/EECS-NTNU/fomo. Please contact the authors to receive access to the Github repository.

8.1 Evaluation Metrics

As explained in the introduction to Chapter 6, the use of simulation is crucial to accurately assess the effect of the heuristic. However, because we solve a multi-criteria optimization problem, it can be difficult to directly compare the quality of different solutions. In general, we want to achieve a high degree of customer satisfaction, meaning a high availability of bikes and locks. Yet, this is affected by both starvations, congestions, roaming and successful bicycle pick-ups and deliveries.

To achieve an efficient and reasonable comparison of policies, we would like an evaluation metric that is simple, yet encompasses all the important aspects. For this purpose, we introduce the concepts of *successful events* and *failed events* to replace the commonly used notion of "trips" in previous research. As failures may occur at both bicycle pick-ups and

returns, we use events to achieve a more nuanced picture.

If users are able to pick up or deliver a bike at their preferred station, it is naturally considered successful events. Starvations, when user demand is lost, are clearly considered failed events. Roaming, on the other hand, can be somewhat more complicated to categorize. The inconvenience of roaming is certainly higher than having demand fulfilled at the desired station, but lower than not having demand met at all. Furthermore, a short roaming distance is better for the user than a long roaming. When a user roams for a bike, the user has already "decided" that the distance is acceptable. Thus, roaming for bikes are considered successful events. This decision making is simulated through the acceptance-rejection method described in Section 6.2.

In contrast, roaming for locks are imposed on users, and should therefor be treated differently. We want to preserve the concept of neighborhood interactions, where nearby stations can absorb demand. However, there is no limit on how far a user may have to roam for a lock, and a long roaming can impose significant user disutility. Therefore, we define a maximum limit for successful roaming for locks. In Chapter 7, we defined a limit of 350 metres for which stations are considered neighbors. The same limit is utilized to differentiate between roaming for locks. This logic is illustrated in Figure 8.1. If a roaming for lock occurs from Station 1, roaming to Stations 2 and 3 are considered successful events, while roaming to Station 4 is not.

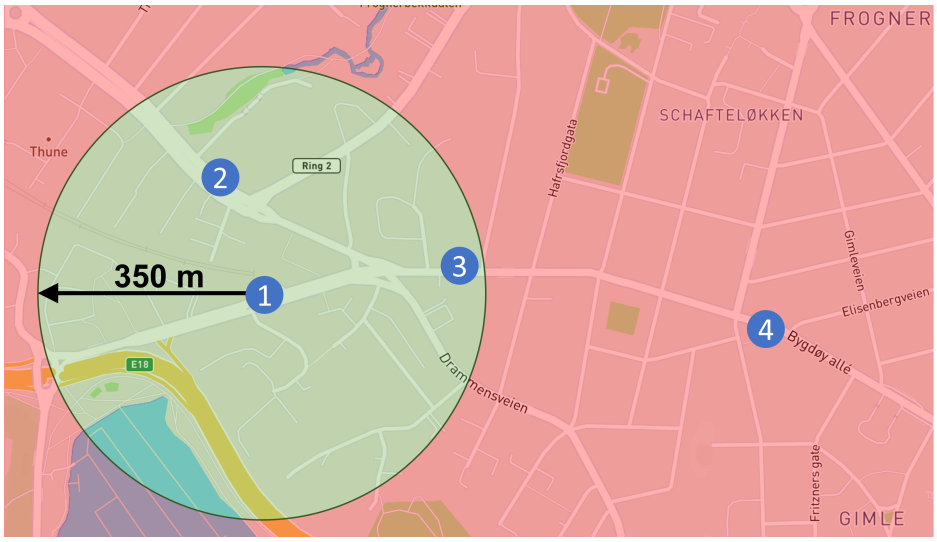


Figure 8.1: Roaming for locks that surpass a given limit are considered failed events due to the considerable inconvenience for the user.

A classification of successful events and failed events is provided in Table 8.2.

Table 8.2: Classification of different types of customer events

Successful events	**Failed events**
Normal bike pick-up	Starvations
Normal bike delivery	Roaming for locks $>$ 350 m
Roaming for bikes	
Roaming for locks $\leq$ 350 m	

When initializing a run in the simulator, the bikes are evenly spread across all stations. The first hours of a simulation are therefore affected by this "perfect" distribution of bikes

and failed events are less likely to occur. To mitigate this weakness, we choose to simulate over at least five consecutive days, every time a simulation is run. Rebalancing operations are performed from 06:00 to 20:00 every day, unless stated otherwise, and the system is open for users between 05:00 and 00:00. This means that the state of the BSS evolves throughout the days and weeks, and the results are less affected by the artificial system balance at the beginning of the simulation.

Even though we simulate several days every time we run a simulation, the results can be vulnerable to randomness, as customer arrivals are generated through Poisson processes. Therefore, when we test a parameter configuration or policy, the simulation is run multiple times on the same instance, with different seeds. However, the same seeds are used for all configurations in order to reduce variability and get more precise results. Our default configuration utilizes 10 seeds, and when necessary, more seeds are added. Finally, the number of failed events is recorded for every simulation run and the final evaluation metric is the average number of failed events over all the runs.

Using this evaluation metric, we achieve a simple comparison between simulation runs that incorporates all the important aspects of the solution quality. However, it should be noted that the solution evaluation may be somewhat dependent on the hard limit of 350 metres that is defined. In addition, there is no differentiation between customer demand being met at the preferred station, and customer demand met through a "successful" roaming. In reality, users are likely to prefer not to roam at all. However, we still believe this to be an adequate evaluation metric for this computational study. It is also worth noting that using the absolute number of failed events as evaluation metric can potentially lead to a bias towards simulation runs with relatively few total events. However, as each simulation is run 10 times, the average number of total events is very similar across different configurations. Preliminary tests in the parameter tuning indicate that using the share of failed events as evaluation metric results in the exact same conclusions as using the absolute number of failed events. Finally, in some situations, it is useful to discuss *service rates* when evaluating the performance of a BSS instance. The service rate is determined by simply dividing the number of successful events by the total number of events.

8.2 Parameter Tuning

As discussed in Chapter 5 and 7, there are several input parameters that affect the performance of the rebalancing algorithm. In this section, a parameter tuning is performed using simulation and the evaluation metrics outlined in Section 8.1. Rather than finding optimal values that are tuned to each test instance, we aim to identify parameter values that are robust and consistently deliver high-quality solutions across multiple test instances. This is done to avoid over-fitting and to ensure a flexible model that is able to adapt to new demand patterns, changes in system characteristics, and new systems. Furthermore, given that simulator data may not perfectly represent real-world data, a more robust and adaptable model is desirable.

When different parameter configurations are compared, they are each run for three different instances. For each instance, we rate the configurations from best to worst according to the evaluation metric from Section 8.1. The best configuration receives a score of one, the second best a score of two, and so on. Then, the scores from the different configurations are summed over all three instances. The configuration with the lowest total score is considered the best, and is therefore selected. By applying this operation, we ensure that

a parameter configuration which performs well for all instances is selected.

The initial parameter configurations are presented in Table 8.3. Parameter values are updated as they are tested during the parameter tuning.

Table 8.3: Initial parameter configuration used in parameter tuning

Initial Parameter Values	
Simulation duration	5 days
Number of seeds	10
Branching depth, α	3
Branching width, β_1	10
Target state, L_i^T	Urban Sharing
Discounting factor, γ_{K_r}	0.8
Criticality weights, ω^{crit}	{(0.4, 0.1, 0.2, 0.2, 0.1), (0.2, 0.4, 0.2, 0.1, 0.1), (0.2, 0.2, 0.1, 0.1, 0.4)}
Time horizon, $\overline{T}$	40 minutes
Number of scenarios	100
Selection criterion	*Consensus*

8.2.1 Evaluation Function Weights

The evaluation function evaluates each plan generated by the construction algorithm. The function looks at three aspects of the plan, *avoided violations*, *enabled roaming*, and *reduced deviation*. These components are described in detail in Section 5.5. Each component is multiplied with a corresponding weight to control their relative importance, as seen in Algorithm 2. These weights always sum to one. Since the returned plan is chosen based on the score it receives from the evaluation, the tuning of these weights can have a significant impact on the performance of the model. In this study, 12 initial weight sets, `a` to `l`, are tested over several test instances using simulation. The aim is to find a configuration of weights that performs well over all instances. A detailed overview of the results is presented in Appendix A.

The results presented in Table 8.4 indicate that multiple weight configurations perform well, but those that prioritize avoided violations yield somewhat lower quantities of failed events. Building on the results of the initial 12 sets, four new weight configurations, `m` to `p`, are created that favor avoided violations. Finally, one of these new configurations is chosen as the default weight set, and is used for the rest of the study. This configuration is marked with green color in Table 8.4.

Table 8.4: Results from computational study on evaluation function weights for avoided violations, ω^v, enabled roaming, ω^r and reduced deviation, ω^d. A lower rank means a better score. The best configuration is highlighted with green color. Detailed results are found in Appendix A

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
ω^v	0.4	0.8	0.1	0.1	0.6	0.3	0.3	0.6	1.0	0.45	0.45	0.33	0.9	0.95	0.85	0.9
ω^r	0.3	0.1	0.8	0.1	0.1	0.6	0.1	0.3	0.0	0.45	0.1	0.33	0.05	0.04	0.1	0.09
ω^d	0.3	0.1	0.1	0.8	0.3	0.1	0.6	0.1	0.0	0.1	0.45	0.33	0.05	0.01	0.05	0.01
$\sum$Rank	27	13	33	48	23	25	44	21	18	25	39	36	12	16	11	17

It should be noted that the the different factors may differ in magnitude. The filtering process described in Section 5.4.1 ensures that all stations visited in a plan deviate from the target inventory level. Consequently, reduced deviation is always a positive value and is typically the largest variable in the evaluation function. In contrast, avoided violations and enabled roaming only take a value when the visited station or some of its neighboring stations are predicted to experience violations. Furthermore, as these values are dependent on the demand during the time horizon, they are typically rather low compared to reduced deviation.

When avoided violations or enabled roaming take a positive value, it indicates that a rebalancing move is likely to have a direct impact on violations. Conversely, reduced deviation, is included in the function to minimize the probability of future violations. Given the higher level of uncertainty associated with future violations, it seems sensible to put a higher emphasis on violations that can be immediately avoided.

8.2.2 Discounting Factors

As explained in Section 5.5, discounting factors are used to capture the relative importance of station visits at different stages in a route. However, it is not trivial to determine what the difference in valuation should be throughout a route, and hence, which discount rate to use. To test different rates, we define γ_{K_r} as the discounting factor for the last visit in a route, r. This means that if γ_{K_r} is set to 0.5, the value of the last visit in a route is scaled down with a factor of 0.5. Based on the number of station visits in route r and γ_{K_r}, an appropriate discount rate can be calculated. Low values of γ_{K_r} translate to high discounting rates, and $\gamma_{K_r} = 1$ gives no discounting.

To find the most appropriate rate, multiple instances are simulated using different values of γ_{K_r}. Values for γ_{K_r} ranging from 0.1 to 1.0 are tested, and the results are presented in Table A.2 in Appendix A.

Simulation results indicate that using γ_{K_r} equal to 0.1 yields the best solution quality, meaning that a high discounting rate is favorable. This is supported by the finding that all γ_{K_r}-values between 0.1 and 0.5 perform better than those between 0.6 and 1. However, despite this trend, there is limited variation in solution quality across the different values of γ_{K_r}, suggesting that the discounting factor has only a modest impact on the final outcome.

8.2.3 Criticality Weights

The rebalancing plans are generated by iteratively adding station visits to the vehicles' routes. Which station to add is determined by the criticality score, which is calculated based on real time information about the system and expected future behavior. Each score consists of five normalized components, each associated with a weight parameter. The components are *Time to violation* (t_i^V), *Deviation from target inventory level* (d_i), *Neighborhood criticality* (n_i), *Net demand* (D_i) and *Driving time* (T_{ji}^D). The calculation of each component is explained in detail in Section 5.4.2.

In order to determine the weight parameter for each component, we follow the procedure outlined in Section 8.2.1. Specifically, we conduct simulations using 12 initial weight sets across various instances. These weight sets are designed to fall under one of three categories: *Balanced*, *Long-term focus*, and *Short-term focus*. Time to violation and deviation from target inventory level have in preliminary tests shown to be important components

for the creation of sensible criticality scores that in turn lead to effective rebalancing. At the same time, these components inherently possess different time scopes. Consequently, within the long-term focus category, the weight assigned to deviation from target inventory level is set at 50% or higher. Similarly, time to violation is weighted with 50% or more when we have a short-term focus. Lastly, in the balanced category, no component is given a weight exceeding 40%.

The goal is to identify the top-performing weight set from each category, and employ them for the remainder of the computational study. Our hypothesis is that incorporating multiple weight sets with different focuses into our model enhances diversification in the planning process. This, in turn, could lead to more robust solution generation, appropriate for different demand patterns throughout the day and various BSS instances. All the weight sets and corresponding results from the simulations are presented in Appendix A.

The results show that the weight sets with short-term focus perform relatively well, with low quantities of failed events across all instances. The best set with a short-term focus is (0.6, 0.1, 0.05, 0.2, 0.05). Conversely, the long-term category, which prioritizes deviation from target inventory level, yields comparatively inferior results. However, one of the weight sets within this category, (0.3, 0.5, 0, 0, 0.2), still performs relatively well across all instances and receives the same rank as the second best set in the short-term category. The weight set that yields the best performance across all instances is located in the balanced category. It consists of the values, (0.3, 0.15, 0.25, 0.2, 0.1), indicating that relatively high weights on time to violation and neighborhood criticality can yield high quality solutions. This set, along with the best sets from the other categories are utilized for the remaining part of this study. The three final weight configurations are presented in Table 8.5.

Table 8.5: Final configurations of weight sets used in the criticality function

	Balanced	**Long-term**	**Short-term**
Time to violation	0.3	0.3	0.6
Deviation from target inventory level	0.15	0.5	0.1
Neighborhood criticality	0.25	0	0.05
Net demand	0.2	0	0.2
Driving time	0.1	0.2	0.05

8.3 X-PILOT Parameter Analysis

In this section we examine the key parameters related to the X-PILOT solution method in terms of solution quality and tractability. These parameters are primarily related to branching and solution selection. While a wider and deeper branching enables a greater exploration of the solution space, it is accompanied by an increase in computational time as a trade-off. As discussed in Chapter 4, the DSBRPNI is an operational problem that must be solved within reasonable time to be practical for real-life operations. To ensure that rebalancing operations are not delayed, we define a limit for *reasonable solution time* to be 10 seconds.

8.3.1 Branching Width & Depth

The branching width β_1 determines the number of potential first-moves that are evaluated. Furthermore, it decides how many successors that are added in each iteration of the X-

PILOT algorithm. Thus, a higher β_1 leads to the generation of more potential solutions. The branching depth α specifies the number of times branching should be performed, and thereby also has a great influence on the size of the X-PILOT tree. An increase in either α or β_1 can enable a greater exploration of the solution space, and possibly solutions of higher quality. However, computational time is expected to increase drastically as more calculations are required. Different combinations of these parameters are tested to examine the trade-off between computational time and solution quality. Only the α and β_1 parameters are changed, while all other parameters are kept constant. Due to the rapid increase in complexity for increasing branching depths when β_1 is high, not all combinations are tested.

The analysis is conducted on the instance `OS_W31_2V` using the parameter values identified in the parameter tuning in Section 8.2. As all service vehicles are integrated into the same X-PILOT trees, the creation of the tree can be influenced by a multi-vehicle set-up. To account for these effects, a two-vehicle instance is used. The time horizon is set to 60 minutes. A full overview of the results are presented in Table B.1 in Appendix B. Figure 8.2 shows how the solution quality varies for different combinations of α and β_1.

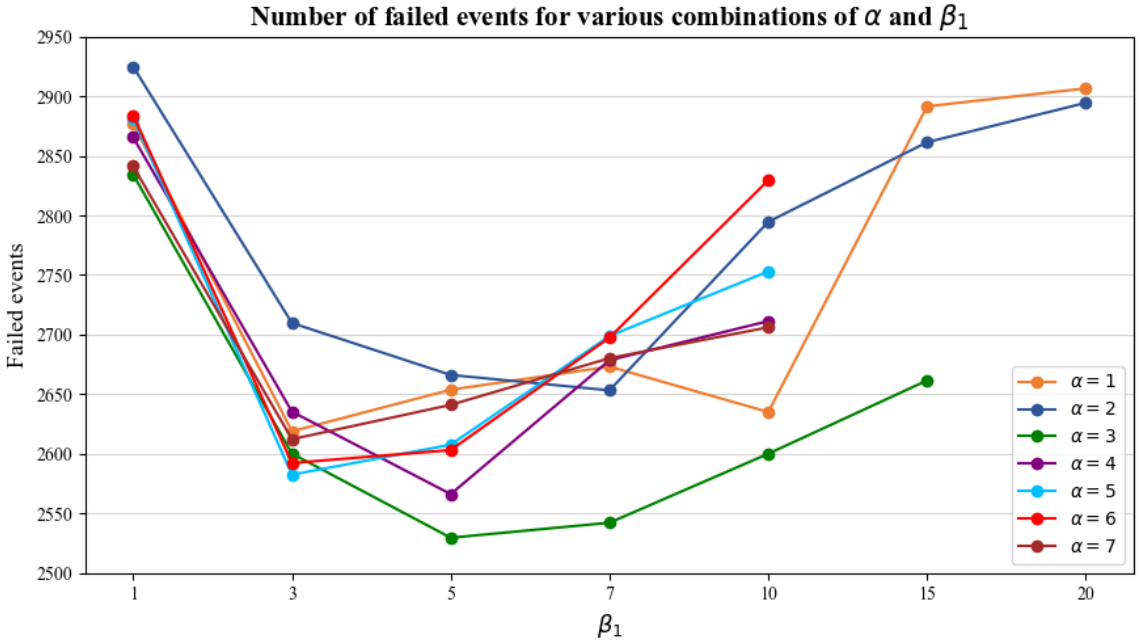


Figure 8.2: Number of failed events for different combinations of α and β_1

Firstly, it is evident from the results that using $\beta_1 = 1$ leads to relatively high numbers of failed events. Across all values of α, the number of failed events for $\beta_1 = 1$ is approximately 2,900 per week. When $\beta_1 = 1$, the algorithm only evaluates the very best station returned from the criticality calculations. Thus, this station is always chosen for the next visit and the algorithm works in a completely greedy manner. It is worth noting that while the figures of approximately 2,900 failed events are the highest recorded in this study, it still accounts for less than 5% of the total events during five days of simulation on the `OS_W31_2V` instance. Recall from Section 7.2 that for this same instance, five days without any rebalancing results in 75.5% successful events. This shows that the greedy approach, which only utilizes the criticality function, performs relatively well. However, it is still possible to improve the solution quality as seen in Figure 8.2. A minor adjustment in the value of β_1, up to three, results in substantially better results. This gives approximately 2,600 failed events, which is a 10% improvement compared to using β_1 equal to one.

Increasing the width further, up to seven, appears to have a negligible impact on the quality as the results are rather similar. Variability in the results makes it difficult to conclude whether it is actually better or not.

An interesting observation, however, is that when β_1 increases even more, the number of failed events tend to increase again. This is somewhat unexpected. When increasing the width of the construction tree, the model should always generate the same plans as for narrower trees, in addition to new plans. The plans in a tree with $\beta_1 = 7$ can be considered a subset of the plans generated with β_1 equal to 10 or 15. Therefore, it is reasonable to expect that the solution quality should be at least as good when increasing β_1. However, this is under the assumption that the evaluation function is perfect and always returns the plan that generates the most utility. When β_1 is increased, stations with lower criticality scores are included in the plans. It is possible that the evaluation algorithm is unable to capture all aspects of a plan, and that it in some cases may return non-optimal first-moves. Increasing the number of possible first-moves beyond a certain point may therefore have a negative impact, as the probability of choosing the most critical stations decreases.

To further investigate this hypothesis, the configuration with $\alpha = 1$ and $\beta_1 = 20$ is run again. Every time rebalancing plans are evaluated for a scenario, we record from which branch the highest evaluated plan originates. Recall that the first branch contains all plans that have the most critical station as their first-move. The second branch, the second most critical station and so forth. Since $\alpha = 1$, there is only generated one plan per branch. Figure 8.3 illustrates the results from these recordings after 5 days of simulation on `OS_W31_2V`.

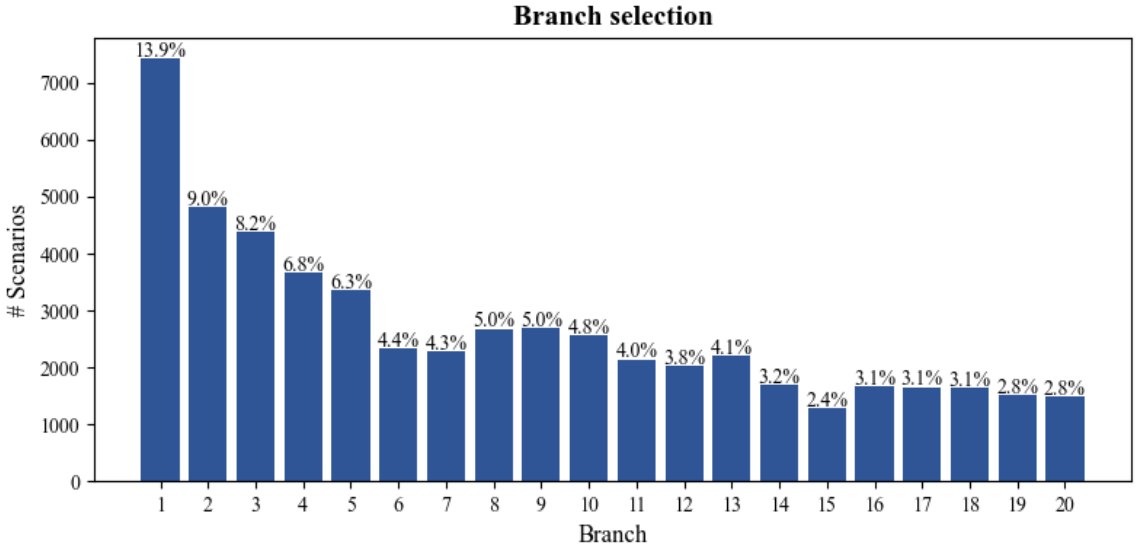


Figure 8.3: Number of times each branch yields the best ranked plan

Unsurprisingly, branch 1, which contains the first-move considered most critical, also returns the rebalancing plan which is evaluated highest across most scenarios. This observation provides reassurance that the evaluation function and criticality function effectively capture many of the same aspects regarding the impact of a station visit. However, we observe that the numbers stabilize after a gradual decrease until branch 5. Despite a small decline, there is not a significant difference in the number of times, for example, branch 6 returns the best plan compared to branch 20. This finding supports the theory that as β_1 becomes excessively large, the evaluation function struggles to differentiate between plans. This can explain why the solution quality deteriorates beyond a certain threshold of β_1.

While this observation holds true, it is evident that the evaluation function makes a positive contribution for lower values of β_1. Decision making based solely on the criticality score yields relatively poor results. However, with just a low number of successors and utilization of the evaluation function, we quickly obtain better results. It is clear that in the construction of the solution tree, a balance between diversification and intensification in the most critical stations must be struck. Opting for a relatively low number of successors and building upon them, appears to yield the best outcomes.

Returning to Figure 8.2, we observe that for different values of α, the trend in solution quality is not equally clear. One observation is that for β_1 equal to one and three, the number of failed events are very similar for most values of α. There is a natural explanation to this. When β_1 is one, the algorithm always constructs plans in a greedy way, and the number of depths does not have any influence on the solution method. A similar argument applies when β_1 is equal to three and five. Recall from Section 5.5 that after the first branching, the β-value is halved for the second and third depth to narrow down the tree in the construction phase. Thus, for $\beta_1 = 3$ and $\beta_1 = 5$, the width is reduced to $\beta_3 = 1$ for the third depth. This means that the configurations with α between two and seven are, in all practical applications, identical. The differences in solution quality stem from uncertainty in the acceptance rejection method.

Figure 8.4 illustrates the increase in solution time as the number of potential first-moves, β_1, increases for different branching depths, α. As expected, the solution time of the algorithm is heavily influenced by the value of these parameters. For fixed α-values, the solution times grow consistently with increasing β_1, although the growth rate varies extensively. For $\beta_1 <= 5$, solution times are below two seconds for all values of α. However, as β_1 is increased from five to seven, solution times begin to grow rapidly, showing the highest growth rate in this study. The growth continues as β_1 is increased to 10, although at a somewhat lower rate.

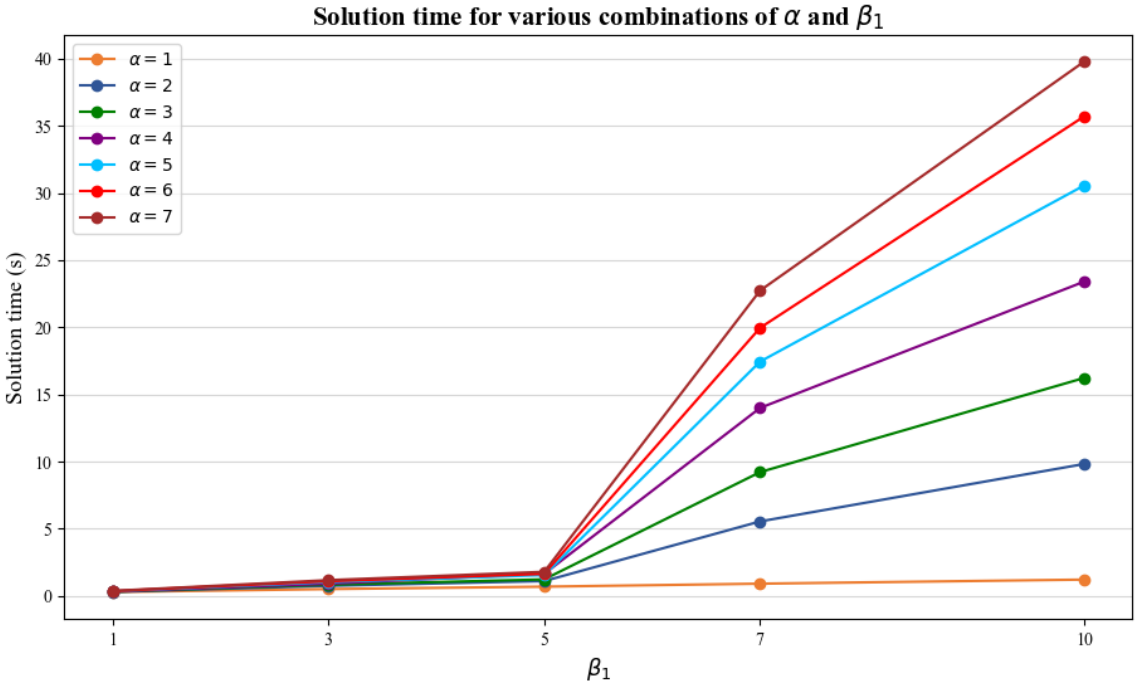


Figure 8.4: Solution time in seconds for different combinations of α and β_1

If we study fixed β_1-values and look at how the solution times changes for varying α, there are some interesting findings. For β_1 lower than five, the solution times are only to a small

extent affected by increasing α. The reason for this is likely also related to the reduction in number of successors. When β_1 is low, the number of successors is quickly reduced to one, meaning that the rest of the construction is done greedily. If this happens before the maximum depth is reached, an even higher depth would not add any benefit as the construction on the last depths is performed greedily either way. Hence, the calculations would be the same, and thereby equally time consuming.

For higher β_1-values on the other hand, the solution time seems to be more sensitive to an increasing α. As we can see from Figure 8.4, the problem cannot be solved within the time limit of 10 seconds when $\beta_1 = 10$ and α is larger than two.

Finally, an interesting observation is that for α-values larger than five, the solution times seem to experience a diminishing increase for high β_1-values. When β_1 is 10, the solution times are all between 30 and 40 seconds for α between five and seven. An explanation can be that, when α is high, the end of the time horizon has already been reached for the plans, before the algorithm utilizes the last depths. When this happens, increasing α does not change anything, as plans are already fully constructed. This is an important observation because it means that the interaction between α, β_1 and the time horizon length has a significant effect on solution times. This should be considered when deciding on parameter values for the model.

8.3.2 Time Horizon Length

The length of the time horizon is a key parameter for the lookahead future of the X-PILOT method. If a longer time horizon is used, more information about the future development of the system is included. This may reduce the probability of myopic decision making. We recall that after the last branching is performed, each plan is completed in a greedy manner until the end of the time horizon. As the greedy construction is rather computationally efficient, the length of the time horizon is only expected to have a minor impact on computational time. However, it is not certain that a more farsighted solution always provides higher solution quality. Therefore, the impact of time horizon length on solution quality and computational time are both subject to analysis. In this study, different time horizons, ranging from 10 to 60 minutes, are tested using simulation on the `OS_W31_2V` instance. For each time horizon, we use three different α and β_1 configurations to capture their interactions with the time horizon. Table 8.6 summarizes the results from the simulation runs.

Table 8.6: Number of failed events and solution time in seconds, for different time horizons, $\overline{T}$. Solution times over 10 seconds are marked in grey.

	α=2, β_1=5		α=3, β_1=10		α=5, β_1=7	
$\overline{T}$	**Failed events**	**Sol. time**	**Failed events**	**Sol. time**	**Failed events**	**Sol. time**
10	3,049	0.37	3,013	0.76	3,021	0.81
20	2,803	0.47	2,895	2.10	2,819	1.95
30	2,561	0.63	2,719	5.27	2,684	4.61
40	2,428	0.80	2,749	8.76	2,724	7.56
50	2,562	0.95	2,633	12.05	2,765	11.28
60	2,609	1.08	2,572	16.05	2,674	16.76

Naturally, as the time horizons lengthen, the solution times also increase. However, the increase remains modest for the first configuration. When α is two and β_1 is five, the solution time rises from 0.37 seconds for a 10-minute time horizon to 1.08 seconds for a

60-minute time horizon. It is worth noting that when the depth is as low as two, most of the construction for longer time horizons is performed greedily. This explains why the solution times remain low even when the time horizon is increased.

However, as α and β_1 increase, the solution times start to exhibit more significant variations. In the second and third configurations (as shown in Table 8.6), the solution times increase from less than a second to over 16 seconds for the longest time horizon. In these cases, the algorithm does not reach its full depth within the shorter time horizons. However, it does when the time horizon is extended. This, coupled with a broader construction tree, can account for the substantial differences in solution times. Finally, for the last two configurations, utilizing time horizons of 50 and 60 minutes results in solution times exceeding our predefined limit of a reasonable time of 10 seconds.

In terms of solution quality, we observe an improvement for longer time horizons. Specifically, the model exhibits its poorest performance when using a time horizon of 10 minutes, with an average of over 3,000 failed events across the three configurations. In contrast, the 60-minute time horizon demonstrates the best average performance, with just over 2,600 failed events. Moreover, it performs best on the second and third configuration, while the 40-minute time horizon excels in the first configuration and has the second-highest average score. Except for the shortest time horizons, the differences in solution quality are limited. This may indicate that there is a significant value to the lookahead feature, but that the value is diminishing for time horizons beyond a certain length.

Considering the desire for a balance between solution quality and time efficiency, a time horizon of 40 minutes seems appropriate. This allows us to achieve satisfactory solution quality while ensuring that the solution time remains relatively short, staying within our defined limit for all the assessed branching configurations.

8.3.3 Number of Scenarios

The X-PILOT solution method employs evaluation of solutions across multiple scenarios as a mean of addressing uncertainty. By testing a solution across multiple scenarios, we can assess its robustness and ensure that it performs well under different circumstances, including those that may be unexpected. However, using many scenarios means that more computational efforts are needed for both scenario generation and for solution evaluation.

As explained in Chapter 5, demand realizations at stations are created by sampling from a Poisson Distribution, using the expected arrival and leave intensities of customers as rates. For comparison, we also examine how the model performs when it evaluates plans based solely on the expected net demand. Through simulation, we examine how the number of scenarios affect solution time and whether it has any impact on solution quality. A time horizon of 40 minutes is used and the branching parameters are $\alpha = 2$ and $\beta_1 = 7$. The instance used is `OS_W34_2V`, which historically has experienced more demand than `OS_W31_2V`. Five days of operation is simulated and rebalancing is only performed between 06:00 and 14:00. These modifications are made to generate more failed events and hopefully more prominent differences between scenario configurations. Finally, simulation is performed 40 times with different seeds for each configuration and the mean values are presented in Table 8.7.

As expected, the solution time increases with the number of scenarios employed. When using a single scenario, the solution time is recorded at 0.97 seconds. With 10 scenarios, the solution time only rises to 1.16 seconds. These results suggest that both scenario gener-

Table 8.7: Number of failed events and solution times in seconds for different numbers of scenarios, and when using expected net demand. Solution times over 10 seconds are marked in grey.

Scenarios	**Failed events**	**Sol. time**
Expected demand	10,609	0.90
1	10,792	0.97
10	10,673	1.16
100	10,630	3.43
500	10,578	13.99
1,000	10,573	27.11
2,000	10,530	53.32

ation and plan evaluation are relatively efficient. Scaling up to 100 scenarios, the solution time extends to 3.43 seconds, comfortably within our predefined limit of a *reasonable time* of 10 seconds.

Beyond 100 scenarios, the solution times appear to increase proportionally to the number of scenarios employed. This can be explained by the fact that with a high scenario count, the processes of scenario generation and route evaluation constitute the majority of the total solution time. The time it takes to construct plans is not affected by an increase in scenarios. An increase to 500 scenarios leads to a solution time of 13.99 seconds, and hence a solution time that by our definition is not applicable for practical use. For 1,000 and 2,000 scenarios, the solution time doubles, further emphasizing the impracticality of such high scenario counts.

It is important to remember that the solution times are dependent on several other parameters, in addition to the number of scenarios. Increasing the time horizon leads to longer routes, and utilizing higher branching parameters leads to more rebalancing plans in the PILOT tree. As a consequence, the time used to evaluate plans is increased. Thus, it is difficult to decide a maximum number of scenarios that can be used for practical applications, without considering all parameters simultaneously.

With regards to solution quality, the number of scenarios used seems to be relatively important. When the plan is evaluated over only one scenario, the results are comparatively poor with an average of 10,792 failed events across the 40 simulations. Using only one scenario, it is random whether the predicted demand in the scenario is close to or far from the later realized demand. Therefore, it is not surprising that this configuration performs poorly. As the number of scenarios increases, there is a noticeable decrease in the occurrence of failed events, although not very large. Specifically, when considering 2,000 scenarios, the number of failed events is reduced to 10,530. This is an improvement of 262 events or 2.4% compared to using one scenario. Still, the decrease seems to be diminishing for higher scenario counts, and the largest improvement is from one to ten scenarios.

An interesting observation is that evaluating plans using only the expected demand yields acceptable results with an average of 10,609 failed events. This approach is the fastest of the alternatives and can be suitable if solution time is decisive. Nevertheless, our analysis demonstrates that incorporating multiple scenarios for evaluation leads to a reduction in failed events. This highlights the significance of addressing the problem's stochastic nature through demand scenario sampling.

In addition to examining the absolute values of failed events for different number of scen-

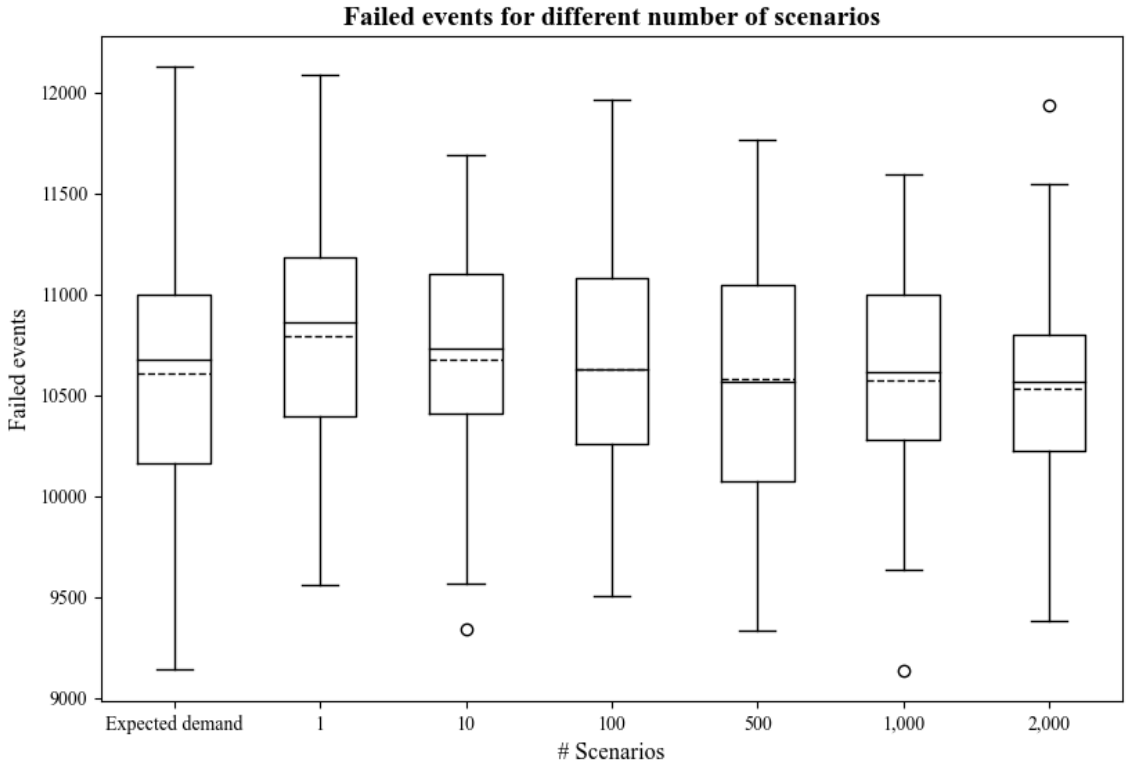


Figure 8.5: Box plot comparing how the number of failed events are dispersed for 40 different simulation runs, across various scenario configurations. Full lines represent the median values, while dashed lines show the means.

arios, it is interesting to address the variability and distribution of the results across the 40 simulations. In Figure 8.5, this is illustrated using a box plot. The full lines represent the median values, while dashed lines show the mean values of failed events. The boxes represent the interquartile ranges (IQR), from the lower to the upper quartiles. This means that the middle 50% of observations are dispersed between these intervals. The end of the whiskers show the extreme values, excluded outliers. Finally, if an observation lies outside 1.5 times the IQR, above the upper quartile or below the lower quartile, it is considered an outlier and is marked with a circle.

Although the evaluation using expected demand yields a relatively good mean value, it also results in the largest spread across the 40 simulations and the highest maximum value, exceeding 12,000 failed events. By utilizing a higher number of scenarios, the variability seems to be reduced to a certain extent. Notably, the configuration with 2,000 scenarios has the shortest IQR, while the configuration with 1,000 scenarios demonstrates the lowest maximum value.

Yet, there are some surprising observations when analyzing the plot. The results from the configuration with 500 scenarios appear to be more dispersed than for 10 and 100 scenarios. In addition, there seems to be relatively high variability across all configurations, with differences of over 2,000 failed events between the maximum and minimum values for each configuration. However, this may be explained by general variations in the simulation results. Table B.2 in Appendix B reveals a distinct pattern between seed numbers and the number of failed events. Certain seeds consistently yield high numbers of failed events across all scenario configurations, while others consistently produce low numbers across all configurations. This indicates that a significant portion of the variability in the

results originates from variations in the simulation processes rather than the evaluation of rebalancing plans.

8.3.4 Selection Criteria

After all plans have been generated and evaluated across multiple scenarios, only one corresponding first-move is selected. However, since we solve a stochastic optimization problem, it may not be possible to identify a single optimal solution. Three different selection criteria are discussed in Section 5.5.2, each with different advantages and drawbacks. To analyze how selection criteria impact the quality and characteristics of solutions, we evaluate two of these approaches.

The first selection criterion we choose to analyze is *consensus*. This approach finds the best plan and corresponding first-move for each scenario, before it selects the first-move that is returned most often. Secondly, tests are performed with the criterion *expectation*. This method identifies the plan with the highest average evaluation score over all scenarios and returns the corresponding first-move. More detailed descriptions of the criteria are found in Section 5.5.2.

A two-sample t-test is conducted to analyze whether there is a statistically significant difference between the results achieved from the two criteria. A thorough description of the t-test is provided in Appendix C. The test is conducted on the instances `OS_W31_2V`, `TD_W34_1V` and `BG_W35_1V` using a significance level of $\alpha = 5\%$ and degrees of freedom $df = 19$. We perform a pairwise comparison of results from the two different criteria, X (*expectation*) and Y (*consensus*), denoting the difference between them $Z = X - Y$. Thus, a negative Z-value means that the *expectation* criterion results in fewer violations. Our null hypothesis states that the two criteria perform equally well, $Z = 0$, whereas the alternative hypothesis is that one criterion outperforms the other, $Z \neq 0$. Different instances are used to see whether the characteristics of the instances have an impact on the performance of the selection criteria. Results are summarized in Table 8.8.

Table 8.8: Results from Two-Factor t-test on two different selection criteria

Results of Two-Sample t-test			
Instance	`OS_W31_2V`	`TD_W34_1V`	`BG_W35_1V`
Sample Mean $\overline{Z}$	-40.2	-8.2	-8.8
95% Confidence Interval	(-122.9, 42.6)	(-14.0, -2.3)	(-22.7, 5.2)

The results reveal that there is in fact a statistically significant difference for the `TD_W34_1V` instance. As zero is not a part of the confidence interval, we can say with a 95% probability that the mean value for the *expectation* criterion is lower than the mean for *consensus*. This indicates that *expectation* provides fewer failed events for the `TD_W34_1V` instance. For the `OS_W31_2V` and `BG_W35_1V` instances, *expectation* also provides a lower mean value. However, there is not statistical foundation to claim a significant difference. Hence, we cannot conclude that there is a performance difference for these two instances.

8.4 Computational Performance

The purpose of this performance analysis is to asses whether the algorithm can solve large instances within a reasonable time for real-world applications. Up until this point, BSSs in

Trondheim, Bergen, and Oslo have been tested with one to two service vehicles, all of which are small to medium-sized instances. This study evaluates the scalability of the solution approach to determine its suitability for solving the DSBRPNI for larger BSSs. Recall that we define *reasonable solution time* as less than 10 seconds. Because tractability is the focus of these test, we only consider solution time, and are not concerned with solution quality.

In order to test our solution method on a larger instance, we here apply the method on the Citi Bike NYC instance. In addition to this large-scale BSS, the study is also conducted on the small-sized Bergen instance and the medium-sized Oslo instance. Each instance is tested using one to five service vehicles. Evaluation is performed across 100 scenarios. The time horizon is set to 40 minutes and in the first runs we use $\alpha = 2$ and $\beta_1 = 5$ as branching parameters.

The results presented in Table 8.9 show that the solution method scales remarkably well for larger instances. Even though the New York BSS is several times larger than Oslo and Bergen BSS, the solution times exhibit great consistency. The `NY_W35_1V` instance is solved in less than a second. Increasing to five vehicles, the solution time only extends to 2.25 seconds, which remains well below our threshold for a reasonable time.

Table 8.9: Solution time in seconds for various BSS instances, when $\alpha = 2$ and $\beta_1 = 5$

	x = Number of vehicles				
	1V	2V	3V	4V	5V
BG_W35_xV	0.56	0.99	1.44	1.85	2.26
OS_W31_xV	0.51	0.78	1.02	1.26	1.57
NY_W31_xV	0.91	1.33	1.65	2.01	2.25

As expected, the New York instances require the longest computational time across most vehicle configurations, although the differences are modest compared to the far smaller Bergen instances. A notable finding, however, is that the medium-sized Oslo instances are solved faster than the smaller Bergen instances for all vehicle setups. While it is difficult to conclude on the exact reasons for this, it is a strong indication that instance sizes have a limited impact on solution times.

Analysis in Section 8.3.1 reveals that the configuration with $\alpha = 2$ and $\beta_1 = 5$ is very computationally efficient, whereas wider and deeper X-PILOT trees require far more time. To explore the potential impact of higher values for α and β_1 on the results, we re-run the tests with $\alpha = 3$ and $\beta_1 = 7$. Solution times are expected to be significantly higher, which can make differences in solution time between the instances more prominent. Results are presented in Table 8.10.

Table 8.10: Solution time in seconds for various BSS instances, when $\alpha = 3$ and $\beta_1 = 7$

	x = Number of vehicles				
	1V	2V	3V	4V	5V
BG_W35_xV	2.01	8.45	27.70	85.62	320.78
OS_W31_xV	1.10	3.42	10.64	32.61	100.81
NY_W31_xV	1.96	4.74	11.14	25.42	63.93

As anticipated, the solution times increase for all instances when the width and depth are extended. We also observe a much steeper development in solution times for increasing number of vehicles. For `OS_W31_xV` the solution time raises from one second for one vehicle,

to 100 seconds for five vehicles. This can be explained by the increase in α and β_1. As we recall from Section 5.5.1, branching also occurs for the vehicles that are not in main focus. With larger depths and widths, the magnitude of branches rapidly grows as the number of vehicles increases.

An interesting observation however is that the smaller Bergen instance requires longer solution times than the instances in Oslo and New York. One explanation for this is that the distances between stations are shorter in Bergen compared to the larger instances. It may therefore be the case that the construction algorithm generates more complex solution trees with larger rebalancing plans within the scope of the time horizon, compared to e.g. New York. In New York, one move can easily make up the entire duration of the time horizon, if the vehicle has to drive across the city. As soon as the time horizon has passed, the algorithm does not add any more station visits to the route, even if it the maximum depth is not reached. Despite this unexpected observation, it is clear that when α and β_1 take higher values, the growth in solution times stems from the increase in number of vehicles and not the number of stations in the BSS.

Because of the way our solution method is implemented, it scales well when the number of stations is increased. Furthermore, when using modest values of α and β_1, which has showed to yield good results, the DSBRPNI can be solved for BSSs with over 900 stations and five service vehicles in less than three seconds.

Chapter 9

Heuristic Performance and Managerial Insights

This chapter provides a comprehensive analysis on the Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions (DSBRPNI), and evaluates the performance of the X-PILOT solution method. In Section 9.1, a comparative study is conducted on X-PILOT and alternative solution methods to assess their performance. X-PILOT features a novel way of coordinating a fleet of service vehicles, and the quality of this coordination is evaluated in Section 9.2. Section 9.3 investigates the incorporation of neighborhood interactions into the rebalancing decisions, exploring their effects and potential benefits. Finally, Section 9.4 discusses the utilization of our performance metric compared to traditional methods, and its implications for managerial decision-making.

9.1 Comparison with Other Policies

In this section we compare the performance of our X-PILOT solution method to alternative approaches for solving the DSBRPNI. First, we simulate the BSSs with no rebalancing operations to establish a baseline. Then, we apply different rebalancing *policies* and compare how these impact the system performance in terms of successful and failed events. By *policy* we here refer to the rules that are used to generate a rebalancing decision based on the available information in a given system state.

9.1.1 Description of Policies

To begin with, we apply an approach which is presented as a benchmark policy by Bakker et al. (2022), here referred to as *Greedy Policy* (GP). This is a semi-greedy approach, in which loading decisions are made greedily in order to get as close as possible to a given target inventory level. The target inventory level is calculated based on expected future demand. Routing decisions are made using filtering and a criticality score. The filtering first excludes stations which are visited by other service vehicles. In addition, either delivery or pick-up stations may be excluded based on the vehicle inventory level. The criticality score takes into account expected demand, driving time, expected time to violation and deviation from target inventory level. The station with the highest criticality score is simply selected.

Another greedy policy that is applied in this comparison is a *Greedy Policy with Neighborhood Interactions* (GPNI). This policy shares several characteristics with the GP. However, a prominent difference is that it takes into account spillover effects between stations. The criticality function from our X-PILOT implementation, including neighborhood criticality, is applied. After calculating the criticality, the station with the highest criticality score is selected greedily. Essentially, this policy is similar to X-PILOT with $\alpha = 1$, $\beta_1 = 1$ and $\overline{T} = 0$, i.e., the X-PILOT policy with no lookahead features.

Further, Kloimüllner et al. (2014) present a PILOT method for the Dynamic Bicycle Rebalancing Problem, here denoted *Kloimüllner PILOT*. The PILOT method presented by Kloimüllner et al. (2014) is a special case of our X-PILOT method, in which the branching depth α is limited to one, and the branching width β_1 is unlimited. This means that the first branching includes all possible stations, and that each branch is then completed in a greedy manner. Note that the implementation of the criticality function and evaluation function used in Kloimüllner et al. (2014) differs from ours. In addition, they create complete routes, off-line, rather than re-solving the problem every time a service vehicle arrives at a station. However, the PILOT-specific parts of their solution method can be adapted to our solution method, even though the implementation is somewhat different. Our implementation gives a best bound on the Kloimüllner PILOT performance.

Finally, our X-PILOT approach is applied, using the parameters identified in the parameter tuning. Branching parameters used are $\alpha = 2$, $\beta_1 = 5$ and $\overline{T} = 40$, which were shown to provide high solution quality within relatively short time in Section 8.3.

9.1.2 Simulation Results

The different rebalancing policies are first simulated on the `OS_W31_2V` instance. Two service vehicles are utilized so that possible differences in coordination between vehicles are accounted for. System operations are simulated for 10 days, with rebalancing being performed from 06:00 to 20:00. The baseline simulations, with no rebalancing operation, results in just more than 30,200 failed events, or a service rate of 71.53%.

Simulation results from the different policies, illustrated in Figure 9.1, show considerable differences between the performance of the policies after 10 days of simulation. Note that the system starts in a balanced state, so the differences only start to show after some time has elapsed. Detailed results are provided in Table D.3 in Appendix D. While the greedy policies exhibit comparatively poor performance, the number of failures is still reduced to 11,350 and 9,878 for GP and GPNI, respectively. This corresponds to a service rate of 89.77% and 91.17%, with GPNI demonstrating the highest performance of the two. Notably, GPNI achieves almost 25% fewer starvations compared to GP. A plausible explanation for this can be that the policy accounting for neighborhood interactions converts starvations into roaming. However, the amount of roaming for bikes is in fact lower for GPNI than for GP, as seen in Table D.3. This points towards a better system balance in terms of distribution of bikes in general.

The two PILOT policies demonstrate higher performance than the greedy policies, with Kloimüllner PILOT resulting in 9,480 failed events, and X-PILOT outperforming the others with only 8,243 failures. Kloimüllner PILOT actually shows a higher number of starvations than GPNI, however, the reduction in long roaming for locks is larger in magnitude. This indicates that the two policies seem to favor different types of violations. X-PILOT, on the other hand, exhibits the lowest number of both starvations and long

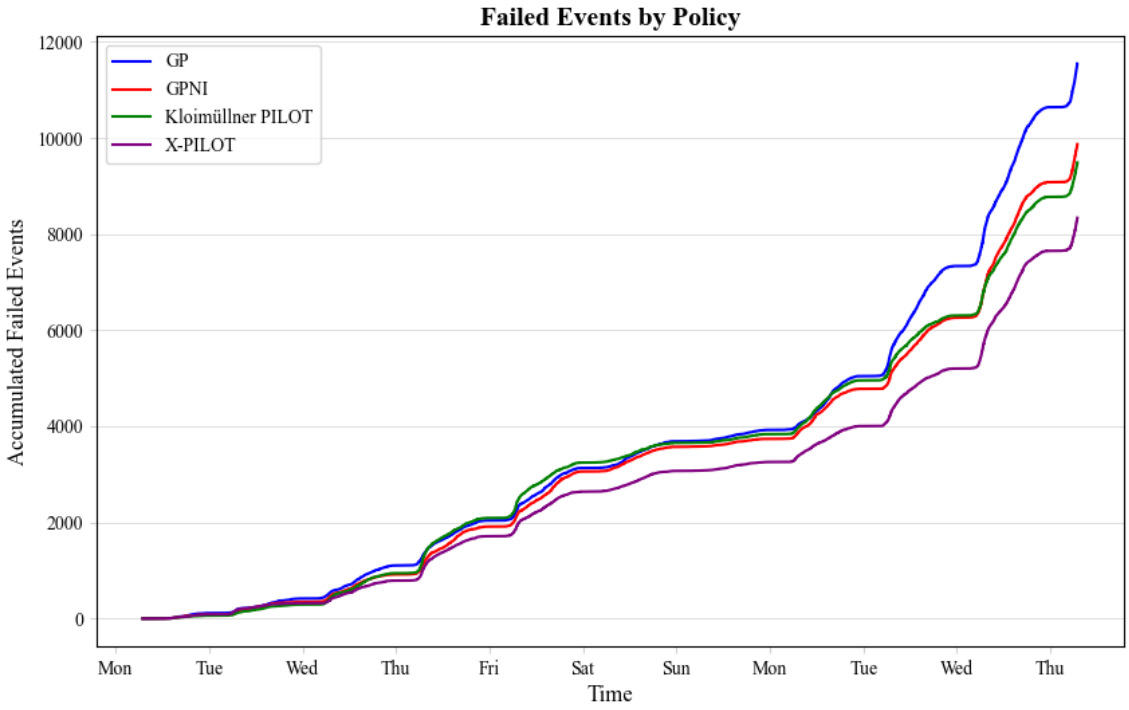


Figure 9.1: Development of failed events for different rebalancing policies for the Oslo instance with two vehicles, plotted for a single simulation

roaming for locks. This results in a service rate of 92.65%.

As noted in Section 9.1.1, the GPNI is effectively the same policy as the X-PILOT without lookahead features. Hence, the performance difference between these can be interpreted as the *value of looking ahead*. Specifically, in this instance, employing lookahead features results in a decrease from 9,878 to 8,243 failed events, representing a notable reduction of 16.6%. This reduction is primarily driven by a decrease in long roaming for locks, accounting for nearly two-thirds of the overall improvement.

Furthermore, the Kloimüllner PILOT can be seen as a special case of X-PILOT, in which all possible first-moves are examined, but no further branching is performed later in the PILOT tree. Therefore, a meaningful part of the difference between the performance of the Kloimüllner PILOT and X-PILOT may be attributed to this additional branching. A deeper branching means that larger parts of the solution space is explored, by evaluating more future options from each first-move. The difference between the two PILOT methods turns out to be rather significant, with the X-PILOT achieving more than 1,200 fewer failed events. Recall that the X-PILOT is solved with $\alpha = 2$, so this additional value is achieved using only one extra depth of branching. However, it should also be noted that the branching width likely affects the results. Studies in Section 8.3.1 reveal that the performance of X-PILOT deteriorates when an excessively wide branching is applied.

Each policy is also simulated for the smaller `BG_W35_1V` instance. This instance is far more self-balanced with approximately 1,900 failures, or a service rate of 96.98% over a 10-day period without any rebalancing efforts. Given the inherently well-balanced nature of this instance, we anticipate observing smaller variations between the policies when applied to it.

The simulation results presented in Figure 9.2, and detailed in Appendix D, confirm that

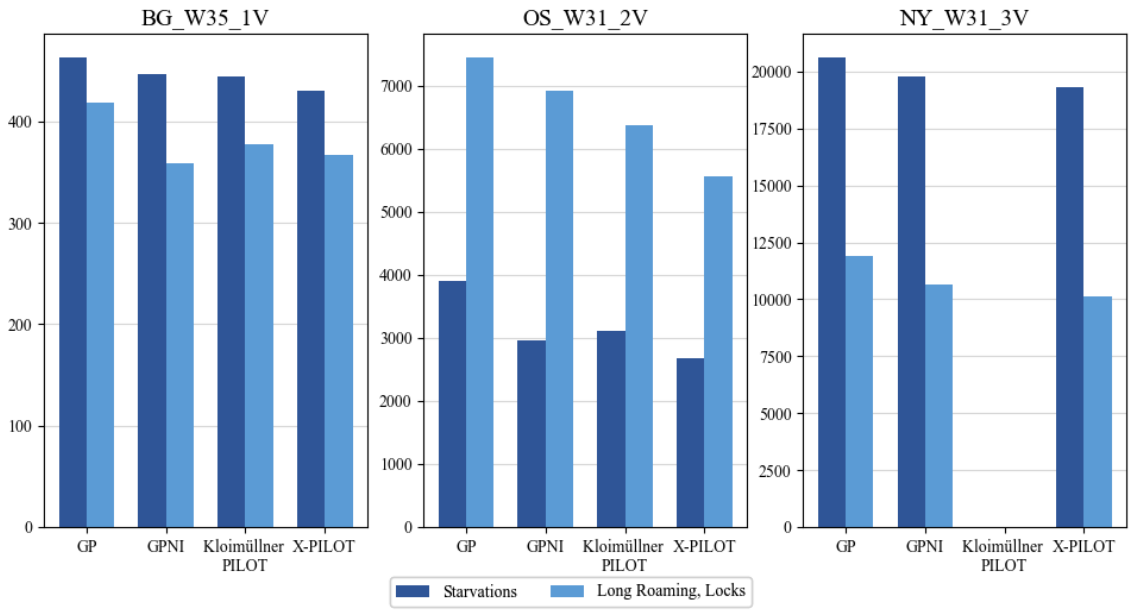


Figure 9.2: Number of starvations and long roaming for locks for different rebalancing policies. Three different instances are tested.

the performance is indeed rather similar between the policies. However, most of the trends observed from the `OS_W31_2V` instance persist. GP exhibits the poorest performance, averaging 881 failures, which is 10.6% more failures than X-PILOT with 796 failed events. X-PILOT continues to demonstrate the best performance, while GPNI and Kloimüllner PILOT are close behind with 807 and 822 failures, respectively.

An intriguing finding is that in the absence of rebalancing, long-distance roaming for locks is by far the most common type of failure, accounting for 71.3% of the total failed events. However, all rebalancing policies cause the distribution of failures to change, with considerably fewer events of long roaming for locks compared to starvations. This observation suggests a higher emphasis on addressing congestions over starvations within the rebalancing policies. This finding gains further support from the analysis of the Oslo-instance, where the reduction in long roaming for locks is notably higher compared to the reduction in starvations.

Next, the large-scale `NY_W31_3V` instance with three service vehicles is simulated for 10 days. Although the system has a high service rate of 95.20% without rebalancing operations, this still represents more than 47,000 failed events, meaning that there is considerable room for improvement by rebalancing.

A notable finding for this large instance is that the Kloimüllner PILOT policy fails to solve the problem within a reasonable timeframe. Consequently, no simulation results are obtained for this particular policy. Recall that the Kloimüllner PILOT policy assesses *all feasible successors*, which can be an extensive set of stations for the New York BSS. This high number of successors is likely the primary reason for the lacking tractability.

Apart from the deficiency of the Kloimüllner PILOT policy, results from the `NY_W31_3V` align with the observations made for the other instances. X-PILOT demonstrates superior performance with 29,942 failed events, followed by GPNI with 30,475 failed events. Once again, the GP policy falls short in matching the performance of the other policies, resulting in a total of 32,552 failed events.

Interestingly, the most significant performance disparities between GP and the other policies arise from long-distance roaming for locks. While the number of starvations shows a 6.6% increase compared to X-PILOT, the occurrence of long roaming for locks is 17.5% higher. A similar trend is observed when comparing with GPNI. One possible explanation for this discrepancy is that both GPNI and X-PILOT take into account the spillover effect and roaming distances between stations, while GP neglects these factors. Consequently, GPNI and X-PILOT value a conversion from a long roaming into a shorter roaming, whereas GP considers both cases as congestion.

In conclusion, the X-PILOT solution method is proven to outperform alternative solution methods across all the analyzed instances. The magnitude of improvement is particularly pronounced in larger and more imbalanced systems, where the potential for enhancement is most significant. This study shows that the selection of rebalancing policy can have a considerable effect on user satisfaction. Although an increase in service rate from 89.77% to 92.65% for the Oslo instance may not seem vast in terms of percentage points, it translates to a reduction of up to 75,000 failed events throughout a season, assuming that the simulated period is representative.

9.2 Coordination of Service Vehicles

Section 9.1 reveals that X-PILOT performs better than other benchmark policies. However, we have not yet examined in detail where the improvements stem from. In Section 5.5.1, we establish that our construction algorithm considers all service vehicles in the system concurrently during the creation of rebalancing plans. This coordinated approach aims to optimize the utilization of current vehicle locations when determining their subsequent destinations. The underlying expectation is that this methodology capitalizes on synergies within a multi-vehicle context, resulting in enhanced overall performance, compared to other approaches.

To examine whether this is the case, we simulate how the policies GP, GPNI and X-PILOT perform with different number of vehicles in Oslo. In real life, four vehicles are employed in the Oslo BSS during periods of high demand. We therefore perform tests on configurations with one to four vehicles in this study. For comparison, the only coordination in GP and GPNI is that vehicles cannot drive to stations where other vehicles are already headed. If the marginal benefit of additional vehicles is greater in X-PILOT than in GP and GPNI, this can indicate that a more coordinated approach is achieved. 10 days of simulation is conducted for each vehicle configuration on the `OS_W31_xV` instance, with rebalancing operations from 06:00 to 14:00. The relatively short rebalancing period is used to avoid the system from becoming too balanced when adding multiple vehicles, making it difficult to compare solution quality.

The results presented in Table 9.1 demonstrate that with the coordinated approach of X-PILOT, we indeed experience a greater reduction in failed events compared to the other policies. When utilizing only one vehicle, the number of failed events are nearly identical across the three polices, approximately 35,000. This translates to a service rate of 79%. However, when increasing to two vehicles, X-PILOT achieves a reduction in failed events to 26,461, an improvement of 8,036 events or 23.3%. In contrast, GP and GPNI exhibit comparatively lower improvements of only 18.6% and 18.5%, respectively.

This trend persists as the number of vehicles increases to three. With X-PILOT, the number of failed events is reduced by 20.9%, whereas GP and GPNI obtain reductions of

16.9% and 19.9%, respectively. Further increasing the number of vehicles to four, GPNI experiences the greatest improvement, while the performance of GP lags behind. X-PILOT demonstrates a slightly lower improvement than GPNI. This can be attributed to the challenge of achieving further enhancements as the service rate approaches a comparably high level. It suggests that the benefits of additional vehicles diminish as the system achieves a more balanced state. This theory finds support in the diminishing improvement in absolute numbers per additional vehicle across all policies.

Table 9.1: Service rate, number of failed events and relative improvement of failed events when employing more vehicles for different policies.

Policy	**# vehicles**	**Service rate**	**Failed events**	**Improvement**	
GP	1	78.7%	35,086	-	-
	2	82.8%	28,560	6,526	18.6%
	3	85.8%	23,748	4,811	16.9%
	4	88.2%	19,882	3,867	16.3%
GPNI	1	79.0%	34,706	-	-
	2	83.1%	28,277	6,429	18.5%
	3	86.5%	22,664	5,613	19.9%
	4	89.5%	17,790	4,874	21.5%
X-PILOT	1	79.1%	34,497	-	-
	2	84.2%	26,461	8,036	23.3%
	3	87.6%	20,922	5,539	20.9%
	4	90.3%	16,488	4,434	21.2%

9.3 Effects of Neighborhood Interactions

This section highlights the integration of neighborhood interactions into the decision-making process for the Dynamic Bicycle Rebalancing Problem. The main objective is to assess the impact of this integration on the effectiveness of rebalancing strategies. To achieve this, we conduct simulations on multiple BSS instances, solving the DSBRPNI using our proposed solution method, X-PILOT, with and without consideration of neighborhood interactions. The differences obtained from these simulations are presented and analyzed in this section. Moreover, to determine the statistical significance of the differences observed between the two approaches, two-sample t-tests are conducted with a significance level of 5%.

When neighborhood interactions are not considered, three adjustments are made to the solution method. Firstly, the weight of the *neighborhood criticality* component is set to zero for all criticality weight sets. Secondly, the weight for *enabled roaming* is set to zero in the evaluation function. Lastly, we recall from Section 5.3 that loading quantities are adjusted if there are starved or congested stations in close proximity. This adjustment is not applicable when neighborhood interactions are ignored.

Both solution approaches utilize $\alpha = 2$, $\beta_1 = 5$ and a time horizon of 40 minutes. The evaluation of plans is performed across 100 scenarios, and the selection criterion applied is *expectation*. Tests are performed on multiple different instances to investigate whether the effects of neighborhood interactions are dependent on the characteristics of the BSS. A 10-day simulation period is applied, and the simulations are run with 20 different seeds. Average results from the simulation runs are presented in Table D.1 in Appendix D.

Table 9.2 summarizes the key findings for the different instances. The results reveal that the value of incorporating neighborhood effects varies between the different instances. Several instances experience improved solution quality from the inclusion of neighborhood interactions, although the improvement is modest. For other instances, there is no significant difference between the two solution approaches. In general, the greatest benefits seem to be achieved for the larger and more imbalanced instances, as both `OS_W31_1V`, `OS_W31_2V`, `OS_W34_2V` and `NY_W31_2V` experience a statistically significant reduction in failed events. The solution quality of the smaller and highly well-balanced instances, `TD_W34_1V` and `BG_W35_1V`, is not notably affected.

Table 9.2: Simulation results showing the effect of incorporating neighborhood interactions into the solution method for different instances

	No interactions		**Neighborhood interactions**		
Instance	**Failed events**	**Service rate**	**Failed events**	**Service rate**	**Statistical significance**
`TD_W34_1V`	279	99.04%	288	99.01%	No
`BG_W35_1V`	782	98.76%	804	98.72%	No
`OS_W31_1V`	17,589	83.95%	17,093	84.41%	Yes
`OS_W31_2V`	8,526	92.39%	8,205	92.68%	Yes
`OS_W34_2V`	18,119	89.35%	17,784	89.54%	Yes
`NY_W31_2V`	33,637	96.61%	33,436	96.63%	Yes

When examining the service rates of `TD_W34_2V` and `BG_W35_2V`, it is not very surprising that the inclusion of neighborhood effects have little influence. Considering the high service rates of 99.01% and 98.72% in these systems, they are already well balanced, and there are likely few areas in the systems with large imbalance. This limits the significance of addressing long-distance roaming. The results indicate that almost all user demand is satisfied either directly or through short roaming. This limits the necessity for both rebalancing operations in general, and for addressing neighborhood interactions in rebalancing.

Figure 9.3 provides a more detailed representation of how the solutions differ. One important observation is that for all instances, a considerable amount of roaming occurs, both for bikes and locks. This highlights the importance of recognizing neighborhood interactions. Upon analyzing the instances with statistically significant differences, it is evident that the majority of the discrepancy stems from a reduction in long-distance roaming for locks. The number of short-distance roaming for locks, however, is almost identical. These findings can indicate that the incorporation of neighborhood interactions reduces the number of unwanted events where users must roam a long distance to return their bike. This result harmonizes with the aim of including the spillover effect. As the two policies are almost identical for the most part, we do not expect there to be fewer empty or starved stations. Rather, the aim is to perform a smart selection of stations and loading quantities so that the distances to nearby available options are reduced.

In contrast to the observed effect on long-distance roaming for locks, a similar effect is not observed for starvations and roaming for bikes. Ideally, the inclusion of neighborhood effects should lead to a conversion of starvations into roaming for bikes. The expectation is that if there are more non-starved stations in close proximity to a starved station, users would be more likely to roam instead of giving up on their attempt to pick up a bike. However, the number of starvations and roaming for bikes remains nearly unchanged between the two policies. This finding suggests that our solution method may not fully

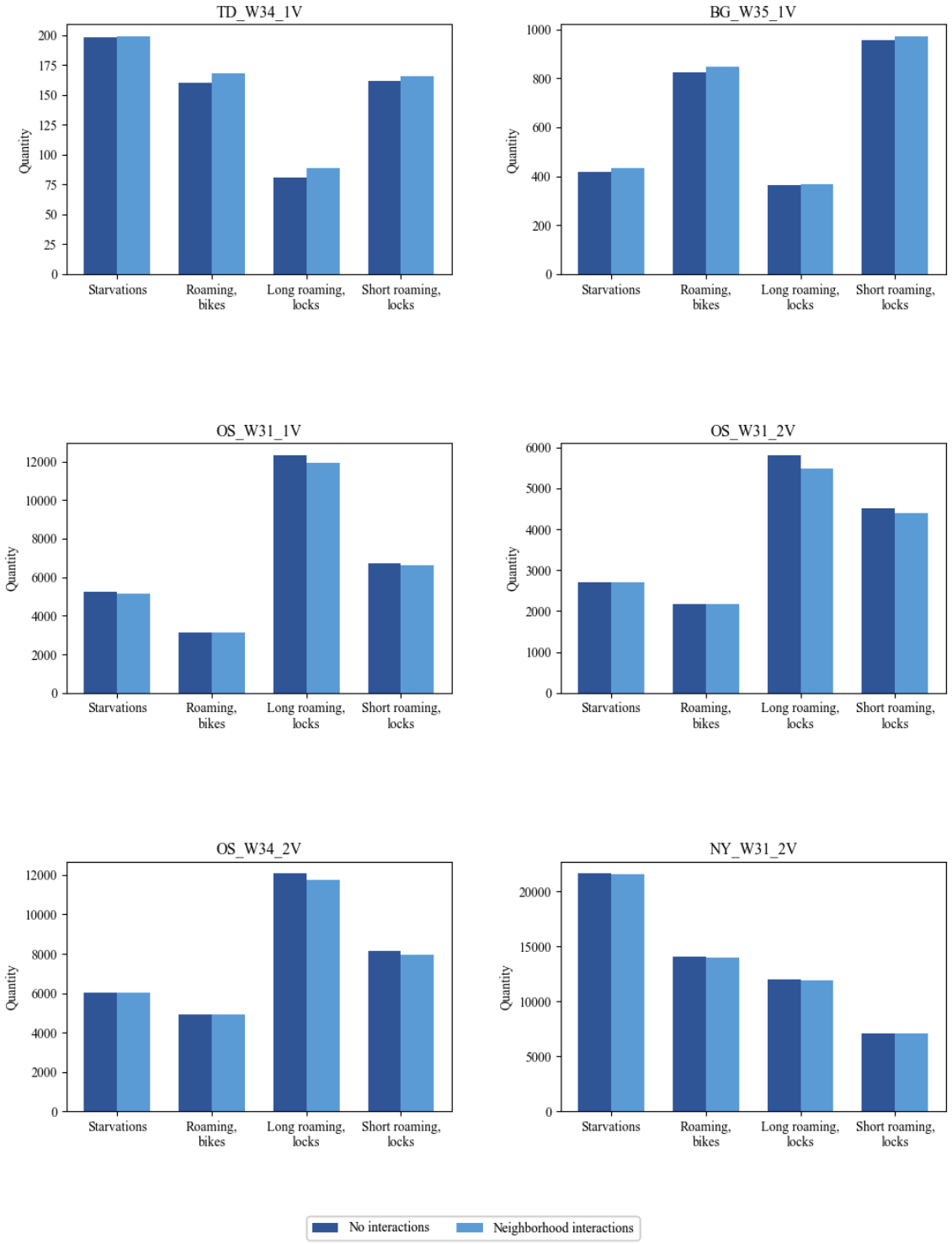


Figure 9.3: How the inclusion of neighborhood interactions in the solution method impacts the performance of different BSS instances.

leverage the spillover effect when it comes to addressing starved stations.

For the instances in Trondheim and Bergen, certain variations can be observed between the two policies. However, it is important to note that the t-tests have confirmed that there is no significant difference between the two solution methods. Therefore, the observed differences are likely to be attributed to natural variations that occur during the simulations. Note that the y-axis in the figure is scaled based on the absolute number of events. As a result, the differences may appear more pronounced when there are fewer events.

9.4 Improved System Performance Metric

Traditionally, the performance of BSSs has been measured in terms of starvations and congestions. A starvation is simply defined as a user arrival at an empty station, and congestion is a bike arrival at a full station. In addition, successes and failures are traditionally defined for trips rather than events. While these measures may provide an indication of the system performance, they fail to include important mechanisms of a BSS and of user utility. In this thesis, we introduce a novel way of measuring system performance and simulating user behavior. This can provide decision makers with better tools to manage BSSs, both in terms of analyzing rebalancing policies and for utilization of resources.

As discussed in Chapter 2, a trip is not necessarily lost once a user arrives at a station. In fact, most user are willing to roam more than 300 metres in order to pick up a bike, as shown in Figure 7.1. The same concept applies to congestions; if a station is full, the user may find available options nearby. Furthermore, each trip consists of both a bicycle pick-up and return, and both events impact user satisfaction. This means that the service rate and user satisfaction cannot simply be captured by looking at stations and trips in isolation, but requires more thorough analysis. We therefore introduce a new event based definition of successes and failures.

It is clear from the study conducted by Costa Affonso et al. (2021) that the likelihood of a lost trip due to a starvation is highly dependent on the distance from the starved station to available options. Thus, this distance should be taken into account. In this thesis, we introduce a roaming module for BSS simulators that mimics actual user behavior, as described in Chapter 6. This allows for more realistic simulations of the system development and more accurate simulation results.

When it comes to congestions, it is reasonable to also here incorporate the effects of roaming distance. For example, if a user can find an available lock within 50 metres it clearly makes little sense to consider this a failed event, as would be the case with traditional evaluation metrics. Instead, we suggest defining a distance limit, so that roaming shorter than this limit is considered a successful event, while longer roaming is considered a failure.

A simulation may yield rather different results, depending on how the performance is measured. This is shown in Table 9.3, where our evaluation metric is compared to a traditional metric for the same simulation. The count of different events is shown for the first seed of our simulation of the `OS_W31_2V` instance used in Section 9.1.

Firstly, a noticeable difference is the number of events or trips. As our metric counts both bicycle pick-ups and returns, the number is almost twice as high. The reason that the number is not exactly doubled is that there is no return-event after a starvation. Further, because traditional metrics only consider starvations, events where the users decide to

Table 9.3: Comparison of Performance Metrics for an Example Simulation

Performance Metric	**Inngjerdingen & Møller**	**Traditional Metrics**
Events/Trips	112,210	57,486
Starvations	2,762	5,070
Roaming, bikes	2,308	-
Short roaming, locks	4,479	-
Long roaming, locks	5,779	-
Congestions	-	10,258
Sum successful	103,669	42,158
Sum failures	8,541	15,328
Service rate	92.4%	73.3%

roam for bikes are also considered starvations. Even if a trip is successfully initiated from a nearby station, the event is noted as a failure. This results in the number of starvations being 83.6% higher when using a traditional metric for this simulation.

Another considerable difference is that our metric does not include congestions. Recall how we assume that a user cannot abandon a bike at a congested station, but has to find a vacant lock elsewhere. This means that there are no "pure" congestions, only roaming for locks. Hence, we distinguish between a long roaming that can be considered a failure, and a short roaming that is of limited inconvenience for the user. This distinction is not present in traditional metrics, meaning that the number of failed events due to congested stations is far higher. For this simulation, the increase is as much as 77.5%.

The resulting service rate from our metric is 92.4%, while the traditional metric yields only 73.3%. If the performance of the system is to be analyzed, this marks a material difference. The magnitude of the difference is of course dependent on the limits used in the definition of a short and long roaming for lock, as well as the probability of a user roaming for a bike. These parameters can be defined by the decision maker according to their preferences and local conditions, but it is clear that the incorporation of neighborhood interactions can be of great significance.

Considering that simulations are often utilized for the development of rebalancing policies and for system management decisions, it is important to have an accurate and realistic performance measure. Thus, we argue that our performance evaluation metric and roaming module may facilitate better rebalancing operations and improved BSS management. As shown in this study, a traditional evaluation method can give a false impression of the service level, and thus might lead to inadequate resource utilization and flawed assessment of rebalancing policies. For example, if the BSS operator in this example aims for a service level of 90%, the conclusion from the simulation results would be very different depending on the metric used. With a service rate of just more than 73%, possible responses could be to add more service vehicles and personnel, or to employ longer shifts. However, if the service rate is over 93%, this would be a waste of resources.

Chapter 10

Concluding Remarks

The main focus of this thesis has been to study the effects of interactions between stations in bike sharing systems (BSSs) in a rebalancing context. A thorough review of existing literature on Dynamic Bicycle Rebalancing Problems (DBRPs) has been performed, which has revealed that limited research has been conducted on the spillover effect between neighboring stations, and how users can roam between stations to fulfill demand. While some research has examined the impact of the spillover effect on demand predictions and the problem of determining the optimal initial inventory level, the DBRP has not yet been addressed while taking this effect into account.

To develop an efficient solution method that can be used for actual decision making in a dynamic setting, we have formulated the Dynamic Stochastic Bicycle Rebalancing Problem with Neighborhood Interactions (DSBRPNI). Furthermore, we have introduced a metaheuristic called the Explorative Preferred Iterative LOokahead Technique (X-PILOT), which is implemented into a solution method for the DSBRPNI. The method incorporates the spillover effect and utilizes lookahead features to make rebalancing decisions while taking future implications into account.

Through a computational study, the X-PILOT method has been proven to scale well, efficiently solving large instances such as Citi Bike New York with more than 900 stations within a few seconds. In terms of solution quality it has been observed that utilizing longer time horizons and more scenarios leads to better results. Nevertheless, the benefits of extending these factors are diminishing after a certain point and must be weighed against the increased solution time. Furthermore, extensive analysis has shown that X-PILOT consistently outperforms other benchmark policies. After 10 days of system simulation, X-PILOT achieved 27.4% fewer failed events compared to a benchmark policy proposed in Bakker et al. (2022). Additionally, when compared to a policy based on the PILOT framework presented by Kloimüllner et al. (2014), X-PILOT exhibited a noteworthy improvement of 13.1%. The lookahead features of X-PILOT were estimated to reduce the number of failed events by 16.6%.

Moreover, our solution method has been shown to perform notably better when more vehicles are employed, compared to other policies that lack vehicle coordination. This suggests that our integrated multi-vehicle approach, which considers all vehicles when constructing solution trees, indeed leads to better coordination and thereby higher service rates.

As simulations are commonly used to analyze BSSs and to develop solution methods for

rebalancing operations, we have developed a roaming module that facilitates more realistic modelling of user behavior and system development. Simulation-based analysis have shown that roaming between stations frequently occurs in BSSs, and that neglecting spillover of demand between stations may thus lead to inaccurate simulations.

Furthermore, we have investigated how the incorporation of roaming in the X-PILOT solution method affects solution quality. The results demonstrated that the effect varies across different instances, and is likely related to the degree of system balance. Well-balanced instances like Trondheim and Bergen showed minimal influence from the incorporation of roaming in decision making. However, larger and more imbalanced instances like Oslo and New York exhibited a statistically significant increase in service rates. These findings highlight the potential benefits of integrating neighborhood interactions in rebalancing strategies, particularly in large and imbalanced BSS instances. Further analysis revealed that the improvement was attributed to a reduction in long-distance roaming for locks, while the number of starvations remained unchanged. This suggests that the X-PILOT method successfully utilizes roaming for locks to enhance system performance, while there is potential for further improvements by incorporating roaming for bikes more effectively.

Finally, we have introduced a novel way of evaluating the performance of BSSs through an improved performance metric that incorporates neighborhood effects into the service rate. While traditional metrics only consider starvations and congestions, neglecting users ability to roam to fulfill demand, we employ a more thorough analysis that considers whether or not user demand would actually be lost, and how severely users are impacted by full stations. We argue that if user demand is easily fulfilled by a nearby station, there is no reason to consider this a failure. Our analysis reveal that traditional metrics tend to greatly underestimate the performance of BSSs by neglecting important aspects of user behavior. We argue that utilizing inaccurate simulations and inadequate performance metrics can lead to inferior system management and waste of resources. Thus, the simulation framework and evaluation metric presented in this thesis may facilitate improved performance for BSS operators.

To conclude, this thesis has shed light on the importance of interactions between stations in bike sharing systems, and shown how the incorporation of these effects can lead to more accurate simulations, better evaluations of system performance, improved rebalancing and better tools for decision making. We have developed a heuristic solution method that can significantly reduce the number of failed events in a BSS, and is well-suited for use in a real-world setting due to short solution times.

Chapter 11

Future Research

This chapter highlights future research opportunities that we believe hold the potential to contribute to the advancement of solution methods for the Bicycle Rebalancing Problem. Moreover, these research avenues offer the prospect of gaining a deeper understanding of important real-world aspects within bike sharing systems. The first section discusses challenges related to understanding the true demand of a BSS. Section 11.2 presents possible improvements of the modelling and understanding of neighborhood interactions. Finally, Section 11.3 discusses other real-life aspects relevant for future studies.

11.1 Demand Censoring

The methodology proposed in this model relies on an accurate estimation of user demand. This estimation plays a crucial role in several key components, including calculating target inventory levels, assessing station criticality, and evaluating rebalancing plans. If the estimated demand fails to provide a reliable approximation of reality, the quality of rebalancing is likely to be unsatisfactory. In this light, it is important to note that the historical data from bike sharing systems only reflects realized demand, not the true demand. Unrealized demand at starved stations is not registered. This concept, referred to as demand censoring, also applies to situations where there is demand for locks at congested stations. Thus, demand is likely to be underestimated at imbalanced stations. As a consequence, the effect of demand spillover between stations is likely to be underestimated as well. With a more realistic representation of demand at imbalanced stations, we would expect the effect of considering neighborhood interactions to be even more prominent. Accurately estimating true demand remains a vital research question that can significantly impact the quality of rebalancing operations.

11.2 Neighborhood Interactions

Even though this thesis introduces several key elements of the interactions between stations, we have identified additional aspects that might further improve the modelling of these effects. We have defined neighborhoods based on the distance between stations, which may not always be an accurate representation of reality. Various obstacles such as rivers or major roads can render nearby stations practically inaccessible, making it infeasible to roam between them. Therefore, it is possible to improve the definition of

neighborhoods to better align with the practical constraints of station connectivity.

Another interesting point is that the nature of roaming may be changing. More and more systems incorporate real-time technology, allowing users to check the availability of stations beforehand. This may eliminate the "traditional" roaming where users arrive at an unavailable station, and initiates roaming from there. Instead, users can identify an available station through and app or web page and go there directly. We argue that this makes roaming easier for users, which may increase the importance of roaming in modern systems. In addition, this might impact the dynamics and user flow in BSS systems, and is an interesting topic for future research.

Furthermore, it is worth exploring how solution methods can be enhanced to better address the issue of starved stations. Results from Chapter 9 indicates that our X-PILOT implementation is able to reduce long-distance roaming for locks, while the number of starvations and roaming for bikes remains unchanged. A more efficient utilization of neighborhood interactions around starved stations can potentially lead to a higher conversion of starvations into roaming for bikes.

11.3 Other Real-Life Aspects

In this section we highlight other real-life aspects that may improve the rebalancing model or be relevant topics for future research. The findings are based on discussions with Urban Infrastructure Partners (UIP) and the authors' own observations from taking part in rebalancing operations in Oslo BSS.

Drivers from UIP presented the issue of varying parking availability throughout the day. Some stations are located in areas where it very challenging to find parking spots during busy hours. For this reason, they are not able to visit these stations at certain periods during the day. Such aspects can be incorporated in rebalancing models in order to ensure feasible solutions. However, gathering the required information can be challenging without local knowledge and experience.

Travel times represent another possible area of improvement. In our model, calculations of travel times, for users and service vehicles, are based on direct distances and fixed speeds. In reality, travelling the direct distance is often not possible, so infrastructure such as roads and bike lanes should be taken into account. Moreover, the travel times often vary throughout the day depending on traffic. Especially in city centers, where BSSs often are located, this can impose significant variations. A solution that might be investigated is integration with online map services with live traffic information, such as Google Maps API.

Another noteworthy research topic is user incentivized rebalancing. Incentivizing users to relocate bikes from congested areas to starved areas can potentially be a useful supplement to traditional rebalancing. As presented in Chapter 3, monetary user incentives is an emerging research topic in BSS literature. UIP managers expressed interest in studies that compare the cost of traditional rebalancing versus user-based rebalancing. The cost per relocated bike can be considerable with vehicle-based rebalancing. An interesting research topic is the comparison of these costs to the level of reward users would require to contribute to rebalancing operations. Even though studies have been conducted using discrete event simulators, analysis of real-life implementations can provide further insights.

Appendix

A Results from Parameter Tuning

Complete results from the parameter tuning performed in Chapter 8 are presented in this appendix. Each combination of configuration and instance is run 10 times, and the average number of total failures are presented in the tables. Based on the number of failures, each configuration is ranked according to the performance relative to other configurations. The best configuration gets a rank of one, the second gets two etc. Then, the rank from each instance is summed for each configuration. This sum is the final evaluation metric, where a lower sum indicates a better performance. The configurations that are selected from the parameter tuning are marked in green. Discussions of results are found in Chapter 8.

A.1 Evaluation Function Weights

The complete results from the tuning of evaluation function weights are presented in Table A.1. Weights are presented in the order (Avoided violations (ω^v), Enabled roaming (ω^r), Reduced deviation (ω^d)). The highest performing configuration, which is selected to be used in the remaining studies, is (0.85, 0.1, 0.05).

A.2 Discounting Factors

Results from the parameter tuning of discounting factors are summarized in Table A.2. Values of γ_{K_r} between 0.1 and 1 are tested, with increments of 0.1. The results indicate that $\gamma_{K_r} = 0.1$ provides the best results, and is hence selected for the following studies.

A.3 Criticality Weights

Results from the parameter tuning of criticality weights are summarized in Table A.3. Weights for the criticality parameters are presented in the order (Time to violation (t_i^v), Deviation from target inventory level (d_i), Neighborhood criticality (n_i), Net demand (D_i), Driving time (T_{ji}^D)). There are four different categories of weights based on their main are of focus; *balanced*, *long term*, *short term*, and *extremes*. The best weight set from each of the first three categories is selected to be used in the remaining studies: (0.3, 0.15, 0.25, 0.2, 0.1), (0.3, 0.5, 0, 0, 0.2) and (0.6, 0.1, 0.05, 0.2, 0.05). The green fields indicate the best scores, and hence the weight sets that are selected from the study.

Table A.1: Results from the computational study on evaluation function weights (ω^v, ω^r, ω^d). The green field highlights the configuration with the best score.

Evaluation weights	Instance	Failed events	Rank	$\sum$ Rank
(0.4, 0.3, 0.3)	OS_W31_2V	2,999	4	27
	TD_W34_1V	151	11	
	BG_W35_1V	364	12	
(0.8, 0.1, 0.1)	OS_W31_2V	2,918	1	13
	TD_W34_1V	112	6	
	BG_W35_1V	286	6	
(0.1, 0.8, 0.1)	OS_W31_2V	3,077	9	33
	TD_W34_1V	154	13	
	BG_W35_1V	355	11	
(0.1, 0.1, 0.8)	OS_W31_2V	3,533	16	48
	TD_W34_1V	168	16	
	BG_W35_1V	406	16	
(0.6, 0.1, 0.3)	OS_W31_2V	3,005	5	23
	TD_W34_1V	128	8	
	BG_W35_1V	347	10	
(0.3, 0.6, 0.1)	OS_W31_2V	3,035	7	25
	TD_W34_1V	129	9	
	BG_W35_1V	325	9	
(0.3, 0.1, 0.6)	OS_W31_2V	3,465	15	44
	TD_W34_1V	164	15	
	BG_W35_1V	379	14	
(0.6, 0.3, 0.1)	OS_W31_2V	3,021	6	21
	TD_W34_1V	124	7	
	BG_W35_1V	305	8	
(1.0, 0.0, 0.0)	OS_W31_2V	3,230	14	18
	TD_W34_1V	56	1	
	BG_W35_1V	263	3	
(0.45, 0.45, 0.1)	OS_W31_2V	3,068	8	25
	TD_W34_1V	134	10	
	BG_W35_1V	305	7	
(0.45, 0.1, 0.45)	OS_W31_2V	3,152	10	39
	TD_W34_1V	158	14	
	BG_W35_1V	379	15	
(0.33, 0.33, 0.33)	OS_W31_2V	3,159	11	36
	TD_W34_1V	153	12	
	BG_W35_1V	364	13	
(0.9, 0.05, 0.05)	OS_W31_2V	2,993	3	12
	TD_W34_1V	93	4	
	BG_W35_1V	271	5	
(0.95, 0.04, 0.01)	OS_W31_2V	3,181	13	16
	TD_W34_1V	68	2	
	BG_W35_1V	252	1	
(0.85, 0.1, 0.05)	OS_W31_2V	2,969	2	11
	TD_W34_1V	102	5	
	BG_W35_1V	267	4	
(0.9, 0.09, 0.01)	OS_W31_2V	3,164	12	17
	TD_W34_1V	71	3	
	BG_W35_1V	262	2	

Table A.2: Results from the computational study on discounting factor γ_{K_r}. The green field highlights the best configuration.

Factor	Instance	Failed events	Rank	$\sum$ Rank
1	OS_W31_2V	2,923	6	17
	TD_W34_1V	91	4	
	BG_W35_1V	271	7	
0.9	OS_W31_2V	2,967	9	26
	TD_W34_1V	95	7	
	BG_W35_1V	277	10	
0.8	OS_W31_2V	2,947	8	26
	TD_W34_1V	98.6	9	
	BG_W35_1V	275.6	9	
0.7	OS_W31_2V	2,925	7	18
	TD_W34_1V	99	8	
	BG_W35_1V	262	3	
0.6	OS_W31_2V	2,885	4	18
	TD_W34_1V	93	6	
	BG_W35_1V	271	8	
0.5	OS_W31_2V	2,916	5	11
	TD_W34_1V	89	2	
	BG_W35_1V	262	4	
0.4	OS_W31_2V	2,979	10	13
	TD_W34_1V	87	1	
	BG_W35_1V	261	2	
0.3	OS_W31_2V	2,817	2	18
	TD_W34_1V	99	10	
	BG_W35_1V	265	6	
0.2	OS_W31_2V	2,825	3	11
	TD_W34_1V	90	3	
	BG_W35_1V	264	5	
0.1	OS_W31_2V	2,753	1	7
	TD_W34_1V	93	5	
	BG_W35_1V	247	1	

Table A.3: Results from the computational study on criticality weights for the parameters $(t_i^v, d_i, n_i, D_i, T_{ji}^D)$. The green fields indicate the best scores.

Criticality weights	Instance	Failed events	Rank	∑ Rank	
(0.2, 0.2, 0.2, 0.2, 0.2)	OS_W31_2V	2,849	10	36	Balanced
	TD_W34_1V	101	13		
	BG_W35_1V	271	13		
(0.3, 0.15, 0.25, 0.2, 0.1)	OS_W31_2V	2,791	5	10	
	TD_W34_1V	85	3		
	BG_W35_1V	254	2		
(0.2, 0.4, 0.2, 0.1, 0.1)	OS_W31_2V	2,761	3	22	
	TD_W34_1V	95	10		
	BG_W35_1V	267	9		
(0.3, 0.3, 0.1, 0.1, 0.2)	OS_W31_2V	2,755	2	20	
	TD_W34_1V	90	8		
	BG_W35_1V	268	10		
(0.2, 0.7, 0.05, 0.05, 0)	OS_W31_2V	2,800	7	32	Long term
	TD_W34_1V	100	11		
	BG_W35_1V	278	14		
(0.05, 0.9, 0.05, 0, 0)	OS_W31_2V	2,884	12	41	
	TD_W34_1V	103	14		
	BG_W35_1V	295	15		
(0.1, 0.6, 0.1, 0.1, 0.1)	OS_W31_2V	2,714	1	19	
	TD_W34_1V	101	12		
	BG_W35_1V	264	6		
(0.3, 0.5, 0, 0, 0.2)	OS_W31_2V	2,792	6	18	
	TD_W34_1V	89	7		
	BG_W35_1V	261	5		
(0.9, 0, 0, 0.1, 0)	OS_W31_2V	3,344	14	22	Short term
	TD_W34_1V	75	1		
	BG_W35_1V	266	7		
(0.7, 0.05, 0.1, 0.1, 0.05)	OS_W31_2V	2,865	11	19	
	TD_W34_1V	86	4		
	BG_W35_1V	259	4		
(0.6, 0.1, 0.05, 0.2, 0.05)	OS_W31_2V	2,785	4	13	
	TD_W34_1V	89	6		
	BG_W35_1V	259	3		
(0.5, 0.05, 0.2, 0.05, 0.2)	OS_W31_2V	2,809	8	18	
	TD_W34_1V	94	9		
	BG_W35_1V	246	1		
(1, 0, 0, 0, 0)	OS_W31_2V	3,289	13	30	Extremes (extras)
	TD_W34_1V	87	5		
	BG_W35_1V	270	12		
(0, 1, 0, 0, 0)	OS_W31_2V	2,830	9	35	
	TD_W34_1V	109	15		
	BG_W35_1V	269	11		
(0, 0, 1, 0, 0)	OS_W31_2V	5,608	16	48	
	TD_W34_1V	218	16		
	BG_W35_1V	541	16		
(0, 0, 0, 1, 0)	OS_W31_2V	5,628	17	51	
	TD_W34_1V	225	17		
	BG_W35_1V	562	17		
(0, 0, 0, 0, 1)	OS_W31_2V	3,383	15	25	
	TD_W34_1V	85	2		
	BG_W35_1V	267	8		

B Results from X-PILOT Parameter analysis

This appendix presents detailed results from the analysis on X-PILOT parameters.

B.1 X-PILOT Width and Depth

In table B.1, it is shown how the solution quality and solution time varies for different combinations of α and β_1. The table presents the average values from 10 simulation runs on the `OS_W31_2V` instance. Simulation is performed over five days and a time horizon of 60 minutes is used. Evaluation is performed over 100 scenarios and the *consensus* criterion is applied.

Table B.1: Number of failed events and solution time in seconds for various combinations of α and β_1. Combinations marked in grey are not tested because of too long solution times.

			β_1						
			1	**3**	**5**	**7**	**10**	**15**	**20**
α	**1**	Failed events	2,877	2,619	2,654	2,673	2,635	2,892	2,907
		Solution time	0.29	0.51	0.69	0.91	1.21	1.73	2.26
	2	Failed events	2,925	2,710	2,666	2,653	2,795	2,861	2,895
		Solution time	0.30	0.75	1.10	5.54	9.82	60.60	136.46
	3	Failed events	2,834	2,600	2,530	2,542	2,600	2,662	
		Solution time	0.31	0.83	1.23	9.22	16.22	333.25	
	4	Failed events	2,866	2,635	2,566	2,679	2,711		
		Solution time	0.32	0.91	1.64	14.00	23.39		
	5	Failed events	2,882	2,582	2,608	2,699	2,753		
		Solution time	0.35	1.03	1.55	17.45	30.55		
	6	Failed events	2,884	2,592	2,603	2,698	2,830		
		Solution time	0.35	1.09	1.64	19.96	35.69		
	7	Failed events	2,841	2,612	2,641	2,680	2,706		
		Solution time	0.37	1.17	1.79	22.73	39.77		

B.2 Number of Scenarios

The X-PILOT solution method employs evaluation of solutions across multiple scenarios as a mean of addressing uncertainty. Table B.2 presents the number of failed events for different scenario configurations, for 40 different seeds.

Table B.2: Number of failed events for different scenario configurations

	Number of scenarios						
Seed	**Exp. demand**	**1**	**10**	**100**	**500**	**1,000**	**2,000**
1	10,656	10,445	10,745	10,317	10,479	10,290	10,449
2	10,396	10,386	10,287	10,546	10,420	10,425	10,455
3	10,737	11,360	10,978	10,928	10,853	10,765	10,397
4	10,313	10,231	9,802	9,911	9,867	9,971	9,783
5	11,464	11,340	11,368	11,028	11,173	11,044	11,230
6	10,313	9,581	9,859	9,793	10,059	10,057	9,583
7	9,714	10,373	10,488	10,056	9,824	10,227	10,155
8	10,934	11,255	11,147	11,166	11,152	10,711	10,836
9	10,106	10,653	10,770	10,506	10,568	10,473	10,642
10	10,829	10,514	10,412	10,750	10,326	10,317	10,466
11	10,166	10,350	10,057	9,927	9,909	10,278	10,320
12	11,011	11,084	10,866	10,770	11,117	10,544	10,511
13	10,219	10,800	11,113	10,592	10,598	10,398	10,305
14	11,289	11,937	11,495	11,343	11,023	11,047	11,180
15	10,333	10,933	10,467	11,077	10,039	10,703	10,801
16	11,206	11,336	10,770	11,277	11,371	11,208	11,350
17	10,310	10,669	10,564	10,508	10,527	10,843	10,644
18	9,994	9,753	9,594	9,884	9,877	9,669	9,827
19	10,058	9,843	10,098	10,156	9,905	9,773	9,979
20	10,698	10,955	11,089	10,381	10,897	11,291	10,620
21	9,140	9,657	9,340	9,586	9,336	9,133	9,383
22	10,072	10,045	10,496	10,294	10,083	10,290	10,184
23	10,882	10,909	11,097	11,197	11,206	10,925	10,792
24	11,559	12,089	11,687	11,297	11,372	11,592	11,938
25	12,129	12,071	11,642	11,965	11,766	11,417	11,546
26	10,450	11,225	10,739	10,582	10,192	10,687	10,358
27	10,801	10,809	10,389	10,535	10,657	10,270	10,240
28	10,863	11,141	10,841	11,090	10,561	10,971	10,800
29	11,292	10,799	11,189	10,881	10,771	10,754	10,745
30	9,491	9,558	9,568	9,504	9,780	9,638	9,456
31	10,992	10,981	11,374	10,837	10,906	11,256	10,725
32	10,056	10,781	10,673	10,508	10,686	10,320	10,015
33	10,589	10,941	10,681	10,659	10,551	10,350	10,684
34	11,103	11,306	10,655	11,170	11,115	11,165	11,105
35	11,070	11,172	10,998	10,852	10,504	10,928	11,014
36	10,808	11,149	10,730	10,862	11,317	10,987	10,658
37	10,031	10,395	10,047	10,083	9,955	10,028	10,101
38	10,691	11,358	11,122	11,109	10,739	11,133	10,671
39	11,444	11,015	11,221	11,145	11,120	11,023	10,917
40	10,157	10,470	10,461	10,117	10,533	10,005	10,324

C Two-Sample t-test

A two-sample t-test is a statistical test which is used to compare the means of two independent groups to determine if they are significantly different from each other. In this thesis, we employ two-factor t-tests to investigate if different solution methods and configurations yield statistically significant differences in solution quality. Since simulation results only offer an approximation of solution quality due to the limited number of seeds simulated, employing statistical tests becomes crucial in assessing if the observed differences in the results are statistically significant. This appendix provides an overview of the two-sample t-tests, offering a general description and outlining their application in our research.

In order to compare two different configurations, X and Y, we do a pairwise comparison of the results obtained from each of the seeds simulated. This means that for each of the n seeds, we set $Z_s = X_s - Y_s$, where X_s and Y_s are the total violations from the two different configurations with seed s. Z_s is the difference between the two results. Employing the t-test, we can assess how likely it is that Z is zero, meaning that the configurations perform equally well for a given significance level α.

We define the following null hypothesis and alternative hypothesis:

- $H_0 : E[Z] = 0$
- $H_1 : E[Z] \neq 0$

where E[Z] is the expected difference between the configurations.

A significance level of α=5% is selected, giving a confidence level of $1 - \alpha = 95\%$. The following steps are conducted in order to create a 95% confidence interval:

- Define sample mean $\overline{Z} = \frac{\sum_{s=1}^{n} Z_j}{n}$
- Define sample variance: $S^2 = \frac{\sum_{s=1}^{n} (Z_s - \overline{Z})^2}{n-1}$
- The variance of the estimate: $\overline{Z}$ is $var(\overline{Z}) = \frac{S^2}{n}$ Confidence interval with confidence level 1-α:

$$\left(\overline{Z} - t_{\frac{\alpha}{2}, n-1} \sqrt{var(\overline{Z})}, (\overline{Z} + t_{\frac{\alpha}{2}, n-1} \sqrt{var(\overline{Z})} \right)$$

The probability that this interval spans the real value of $Z_s = X_s - Y_s$ is $1 - \alpha = 95\%$. Hence, if zero is included in the interval we fail to reject the null hypothesis and conclude that there is not enough evidence to support a significant difference.

An example is provided in Table C.1 for two different selection criteria, using $n = 20$ seeds (see Section 8.3.4). The result is a 95% confidence interval 95% CI [-14.03, -2.27]. As zero is not a part of this interval, we reject the null-hypothesis. Thus, we can say with 95% probability that the different selection criteria provide different solutions.

Table C.1: Results and parameter values for two-sample t-test

Results of Two-Sample t-test	
Sample mean $\overline{Z}$	-8.15
Sample variance S^2	157.92
Var estimate $var(\overline{Z})$	7.90
$t_{\frac{\alpha}{2},n-1}$	2.093
CI lower bound	-14.03
CI upper bound	-2.27

D Results from Heuristic Performance and Managerial Insights

Results from the simulations performed in Chapter 9 are presented in this appendix. Each simulation is run with 20 seeds, and the average numbers are provided here.

D.1 Effect of Neighborhood Interactions

Complete results from simulations of the X-PILOT with and without consideration of neighborhood interactions are presented in Table D.1. This includes each type of event, in addition to service rate and average roaming distances.

Table D.1: How the incorporation of neighborhood interactions in the X-PILOT solution method impacts solution quality for six different BSS instances. In four of the instances, the incorporation yields significantly lower mean values of failed events.

No interactions						
	TD_W34_1V	BG_W35_1V	OS_W31_1V	OS_W31_2V	OS_W34_2V	NY_W31_2V
Starvations	198	417	5,247	2,709	6,050	21,607
Roaming, bikes	160	824	3,135	2,162	4,926	14,049
Long roaming, locks	81	365	12,343	5,816	12,069	12,030
Short roaming, locks	162	958	6,727	4,514	8,156	7,094
Failed events	279	782	17,589	8,526	18,119	33,637
Service rate	99.04%	98.76%	83.95%	92.39%	89.35%	96.61%
Avg. roam dist., bikes	0.28 km	0.21 km	0.31 km	0.30 km	0.29 km	0.33 km
Avg. roam dist., locks	0.34 km	0.27 km	0.46 km	0.42 km	0.44 km	0.43 km
Neighborhood interactions						
	TD_W34_1V	BG_W35_1V	OS_W31_1V	OS_W31_2V	OS_W34_2V	NY_W31_2V
Starvations	199	434	5,166	2,713	6,037	21,526
Roaming, bikes	168	849	3,116	2,177	4,922	13,949
Long roaming, locks	89	369	11,926	5,493	11,747	11,911
Short roaming, locks	166	972	6,606	4,404	7,966	7,081
Failed events	288	804	17,093	8,205	17,784	33,436
Service rate	99.01%	98.72%	84.41%	92.68%	89.54%	96.63%
Avg. roam dist., bikes	0.29 km	0.21 km	0.31 km	0.30 km	0.29 km	0.33 km
Avg. roam dist., locks	0.35 km	0.27 km	0.46 km	0.42 km	0.44 km	0.43 km

D.2 Comparison with Other Policies

Results from the comparison of different rebalancing policies are presented in this section. Table D.2 provides results from the BG_W35_1V instance, while Table D.3 contains results

from `OS_W31_2V`. Results from `NY_W31_3V` are presented in Table D.4.

Table D.2: Results of comparison between policies for the BG_W35_1V instance

Instance BG_W35_1V				
Policy	**Starvations**	**Long roaming, locks**	**Failed events**	**Service rate**
Do nothing	853	1,034	1,887	96.98%
GP	463	418	881	98.60%
GPNI	447	359	807	98.72%
Kloimüllner PILOT	444	377	822	98.69%
X-PILOT	430	367	796	98.73%

Table D.3: Results of comparison between policies for the OS_W31_2V instance

Instance OS_W31_2V				
Policy	**Starvations**	**Long roaming, locks**	**Failed events**	**Service rate**
Do nothing	8,716	21,512	30,228	71.53%
GP	3,908	7,442	11,350	89.77%
GPNI	2,960	6,918	9,878	91.17%
Kloimüllner PILOT	3,117	6,363	9,480	91.51%
X-PILOT	2,680	5,563	8,243	92.65%

Table D.4: Results of comparison between policies for the NY_W31_3V instance. Data for Kloimüllner PILOT is not

Instance NY_W31_3V				
Policy	**Starvations**	**Long roaming, locks**	**Failed events**	**Service rate**
Do nothing	28,665	18,717	47,381	95.20%
GP	20,623	11,929	32,552	96.73%
GPNI	19,810	10,665	30,475	96.94%
Kloimüllner PILOT	-	-	-	-
X-PILOT	19,342	10,149	29,491	97.04%

Bibliography

Andersson, H., Hoff, A., Christiansen, M., Hasle, G., & Løkketangen, A. (2010). Industrial aspects and literature survey: Combined inventory management and routing. *Computers & Operations Research*, *37*, 1515–1536. doi:10.1016/j.cor.2009.11.009.

Bakker, S. J., Djupdal, A., Natvig, L., Andersson, H., Fagerholt, K., & Jahre, M. (2022). *FOMOSim: Enabling rigorous performance evaluation of shared mobility systems using an open-source simulator*. Working Paper. URL: https://github.com/EECS-NTNU/fomo.

Brinkmann, J., Ulmer, M. W., & Mattfeld, D. C. (2020). The multi-vehicle stochastic-dynamic inventory routing problem for bike sharing systems. *Business Research*, *13*, 69–92. doi:10.1007/s40685-019-0100-z.

Büttner, J., Mlasowsky, H., Birkholz, T., Gröper, D., Fernandéz, A., Emberger, G., Petersen, T., Robért, M., Vila, S., Reth, P., Blümel, H., Rodriguez, C., Pineda, E., Piotrowic, A., Ejsmont, R., Kuropatwiski, P., Kowalenska, M., Vecchiotti, F., Reiterer, H., & ... Banfi, M. (2011). *Optimising bike sharing in European cities-a handbook*. OBIS.

Chandler, M. (2020). *The History of Bike Sharing Schemes (And What They'll Look Like In The Future).* Discerning Cyclist. URL: https://discerningcyclist.com/history-of-bike-sharing-schemes-future-predictions/.

Chemla, D., Meunier, F., Pradeau, T., Calvo, R. W., & Yahiaoui, H. (2013). Self-service bike sharing systems: simulation, repositioning, pricing, . URL: https://www.researchgate.net/publication/258222799_Self-service_bike_sharing_systems_simulation_repositioning_pricing.

Chen, F., Turoń, K., Kłos, M., Pamuła, W., Sierpiński, G., & Czech, P. (2018). Fifth generation of bike-sharing systems-examples of poland and china. *Scientific Journal of Silesian University of Technology. Series Transport*, *99*, 05–13. doi:10.20858/sjsutst.2018.99.1.

Chiariotti, F., Pielli, C., Zanella, A., & Zorzi, M. (2018). A dynamic approach to rebalancing bike-sharing systems. *Sensors*, *18*, 512. doi:10.3390/s18020512.

Citi Bike NYC (2023). *Get to know Citi Bike.* Citi Bike NYC. URL: https://citibikenyc.com/how-it-works.

Coelho, L. C., Cordeau, J.-F., & Laporte, G. (2014). Thirty years of inventory routing. *Transportation Science*, *48*, 1–19. doi:10.1287/trsc.2013.0472.

Contardo, C., Morency, C., & Rousseau, L.-M. (2012). Balancing a dynamic public bike-sharing system, . *4*. URL: https://www.cirrelt.ca/documentstravail/cirrelt-2012-09.pdf.

Costa Affonso, R., Couffin, F., & Leclaire, P. (2021). Modelling of user behaviour for static rebalancing of bike sharing system: Transfer of demand from bike-shortage stations to neighbouring stations. *Journal of Advanced Transportation*, *2021*. doi:10.1155/2021/8825521.

Datner, S., Raviv, T., Tzur, M., & Chemla, D. (2019). Setting inventory levels in a bike sharing network. *Transportation Science*, *53*, 62–76. doi:10.1287/trsc.2017.0790.

Delli, K. (2023). Motion for a resolution on developing an EU cycling strategy. URL: https://www.europarl.europa.eu/doceo/document/B-9-2023-0102_EN.html.

DeMaio, P., Yu, C., O'Brien, O., Rabello, R., Chou, S., & Benicchio, T. (2021). The meddin bike-sharing world map: Mid-2021 report, . URL: https://bikesharingworldmap.com/reports/bswm_mid2021report.pdf.

Di Gaspero, L., Rendl, A., & Urli, T. (2013a). Constraint-based approaches for balancing bike sharing systems. In *Principles and Practice of Constraint Programming: 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings 19* (pp. 758–773). Springer. doi:10.1007/978-3-642-40627-0_56.

Di Gaspero, L., Rendl, A., & Urli, T. (2013b). A hybrid aco+ cp for balancing bicycle sharing systems. In *Hybrid Metaheuristics: 8th International Workshop, HM 2013, Ischia, Italy, May 23-25, 2013. Proceedings 8* (pp. 198–212). Springer. doi:10.1007/978-3-642-38516-2_16.

Espegren, H. M., & Kristianslund, J. (2016). *Optimal Repositioning in Bike Sharing Systems*. Master's thesis Norwegian University of Science and Technology.

Faghih-Imani, A., & Eluru, N. (2016). Incorporating the impact of spatio-temporal interactions on bicycle sharing system demand: A case study of new york citibike system. *Journal of transport geography*, *54*, 218–227. doi:10.1016/j.jtrangeo.2016.06.008.

Forma, I. A., Raviv, T., & Tzur, M. (2015). A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation Research Part B: Methodological*, *71*, 230–247. doi:10.1016/j.trb.2014.10.003.

Garcia-Gutierrez, J., Romero-Torres, J., & Gaytan-Iniestra, J. (2014). Dimensioning of a bike sharing system (bss): a study case in nezahualcoyotl, mexico. *Procedia-Social and Behavioral Sciences*, *162*, 253–262. doi:10.1016/j.sbspro.2014.12.206.

García-Palomares, J. C., Gutiérrez, J., & Latorre, M. (2012). Optimizing the location of stations in bike-sharing programs: A gis approach. *Applied Geography*, *35*, 235–246. doi:10.1016/j.apgeog.2012.07.002.

Ghosh, S., Varakantham, P., Adulyasak, Y., & Jaillet, P. (2017). Dynamic repositioning to reduce lost demand in bike sharing systems. *Journal of Artificial Intelligence Research*, *58*, 387–430. doi:10.1613/jair.5308.

Gleditsch, M. D., Hagen, K., Andersson, H., Bakker, S. J., & Fagerholt, K. (2022). A column generation heuristic for the dynamic bicycle rebalancing problem. *European Journal of Operational Research*, . doi:10.1016/j.ejor.2022.07.004.

Hagen, K., & Gleditsch, M. D. (2018). *A column generation heuristic for the dynamic rebalancing problem in bike sharing systems*. Master's thesis NTNU.

Haider, Z., Nikolaev, A., Kang, J. E., & Kwon, C. (2018). Inventory rebalancing through pricing in public bike sharing systems. *European Journal of Operational Research*, *270*, 103–117. doi:10.1016/j.ejor.2018.02.053.

Heineke, K., Kloss, B., & Scurtu, D. (2020). *The future of micromobility: Ridership and revenue after a crisis.* McKinsey & Company. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-future-of-micromobility-ridership-and-revenue-after-a-crisis.

Inngjerdingen, C., & Møller, S. (2022). *Modelling Spillover Effects in the Dynamic Bicycle Rebalancing Problem*. Project Thesis, Norwegian University of Science and Technology.

Jiang, H., & Jamba, H. (2019). *To Solve China's Bike-Sharing Woes, Hangzhou and Shanghai Turn to Bluetooth and Geofencing.* The City Fix. URL: https://thecityfix.com/blog/solve-chinas-bike-sharing-woes-hangzhou-shanghai-turn-bluetooth-geofencing-hui-jiang-harshita-jamba/.

Kaspi, M., Raviv, T., & Tzur, M. (2014). Parking reservation policies in one-way vehicle sharing systems. *Transportation Research Part B: Methodological*, *62*, 35–50. doi:10.1016/j.trb.2014.01.006.

Kaspi, M., Raviv, T., Tzur, M., & Galili, H. (2016). Regulating vehicle sharing systems through parking reservation policies: Analysis and performance bounds. *European Journal of Operational Research*, *251*, 969–987. doi:10.1016/j.ejor.2015.12.015.

Kloimüllner, C., Papazek, P., Hu, B., & Raidl, G. R. (2014). Balancing bicycle sharing systems: an approach for the dynamic case. In *Evolutionary Computation in Combinatorial Optimisation: 14th European Conference, EvoCOP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 14* (pp. 73–84). Springer. doi:10.1007/978-3-662-44320-0_7.

Lin, J.-R., & Yang, T.-H. (2011). Strategic design of public bicycle sharing systems with service level constraints. *Transportation research part E: logistics and transportation review*, *47*, 284–294. doi:10.1016/j.tre.2010.09.004.

Ma, G., Zhang, B., Shang, C., & Shen, Q. (2021). Rebalancing stochastic demands for bike-sharing networks with multi-scenario characteristics. *Information Sciences*, *554*, 177–197. doi:10.1016/j.ins.2020.12.044.

Oslo City Bike (2023). *About Oslo City Bike.* Oslo City Bike. URL: https://oslobysykkel.no/en/about.

Pfrommer, J., Warrington, J., Schildbach, G., & Morari, M. (2014). Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems*, *15*, 1567–1578. doi:10.1109/TITS.2014.2303986.

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*, 1–11. doi:10.1016/j.ejor.2012.08.015.

Puterman, M. L. (1990). Markov decision processes. *Handbooks in operations research and management science*, *2*, 331–434. doi:10.1016/S0927-0507(05)80172-0.

Rainer-Harbach, M., Papazek, P., Hu, B., & Raidl, G. R. (2013). Balancing bicycle sharing systems: A variable neighborhood search approach. In *Evolutionary Computation in Combinatorial Optimization: 13th European Conference, EvoCOP 2013, Vienna, Austria, April 3-5, 2013. Proceedings 13* (pp. 121–132). Springer. doi:10.1007/978-3-642-37198-1_11.

Rainer-Harbach, M., Papazek, P., Raidl, G. R., Hu, B., & Kloimüllner, C. (2015). Pilot, grasp, and vns approaches for the static balancing of bicycle sharing systems. *Journal of Global Optimization*, *63*, 597–629. doi:10.1007/s10898-014-0147-5.

Raviv, T., & Kolka, O. (2013). Optimal inventory management of a bike-sharing station. *Iie Transactions*, *45*, 1077–1093. doi:10.1080/0740817X.2013.770186.

Regue, R., & Recker, W. (2014). Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem. *Transportation Research Part E: Logistics and Transportation Review*, *72*, 192–209. doi:10.1016/j.tre.2014.10.005.

Romero, J. P., Ibeas, A., Moura, J. L., Benavente, J., & Alonso, B. (2012). A simulation-optimization approach to design efficient systems of bike-sharing. *Procedia-Social and Behavioral Sciences*, *54*, 646–655. doi:10.1016/j.sbspro.2012.09.782.

Ruch, C., Warrington, J., & Morari, M. (2014). Rule-based price control for bike sharing systems. In *2014 European Control Conference (ECC)* (pp. 708–713). IEEE. doi:10.1109/ECC.2014.6862386.

Rudloff, C., & Lackner, B. (2014). Modeling demand for bikesharing systems: neighboring stations as source for demand and reason for structural breaks. *Transportation Research Record*, *2430*, 1–11. doi:10.3141/2430-01.

Russell Meddin (2022). *The Meddin Bike-sharing World Map*. URL: https://bikesharingworldmap.com/.

Schuijbroek, J., Hampshire, R. C., & Van Hoeve, W.-J. (2017). Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, *257*, 992–1004. doi:10.1016/j.ejor.2016.08.029.

Shaheen, S., Guzman, S., & Zhang, H. (2010). Bikesharing in europe, the americas, and asia: Past, present, and future. *Institute of Transportation Studies, UC Davis, Institute of Transportation Studies, Working Paper Series*, *2143*. doi:10.3141/2143-20.

Shui, C., & Szeto, W. (2020). A review of bicycle-sharing service planning problems. *Transportation Research Part C: Emerging Technologies*, *117*, 102648. doi:10.1016/j.trc.2020.102648.

Szeto, W., Liu, Y., & Ho, S. C. (2016). Chemical reaction optimization for solving a static bike repositioning problem. *Transportation research part D: transport and environment*, *47*, 104–135. doi:10.1016/j.trd.2016.05.005.

Szeto, W. Y., & Shui, C. S. (2018). Exact loading and unloading strategies for the static multi-vehicle bike repositioning problem. *Transportation Research Part B: Methodological*, *109*, 176–211. doi:10.1016/j.trb.2018.01.007.

Urban Sharing (2023). *Making Micromobility Profitable.* Urban Sharing. URL: https://urbansharing.com/.

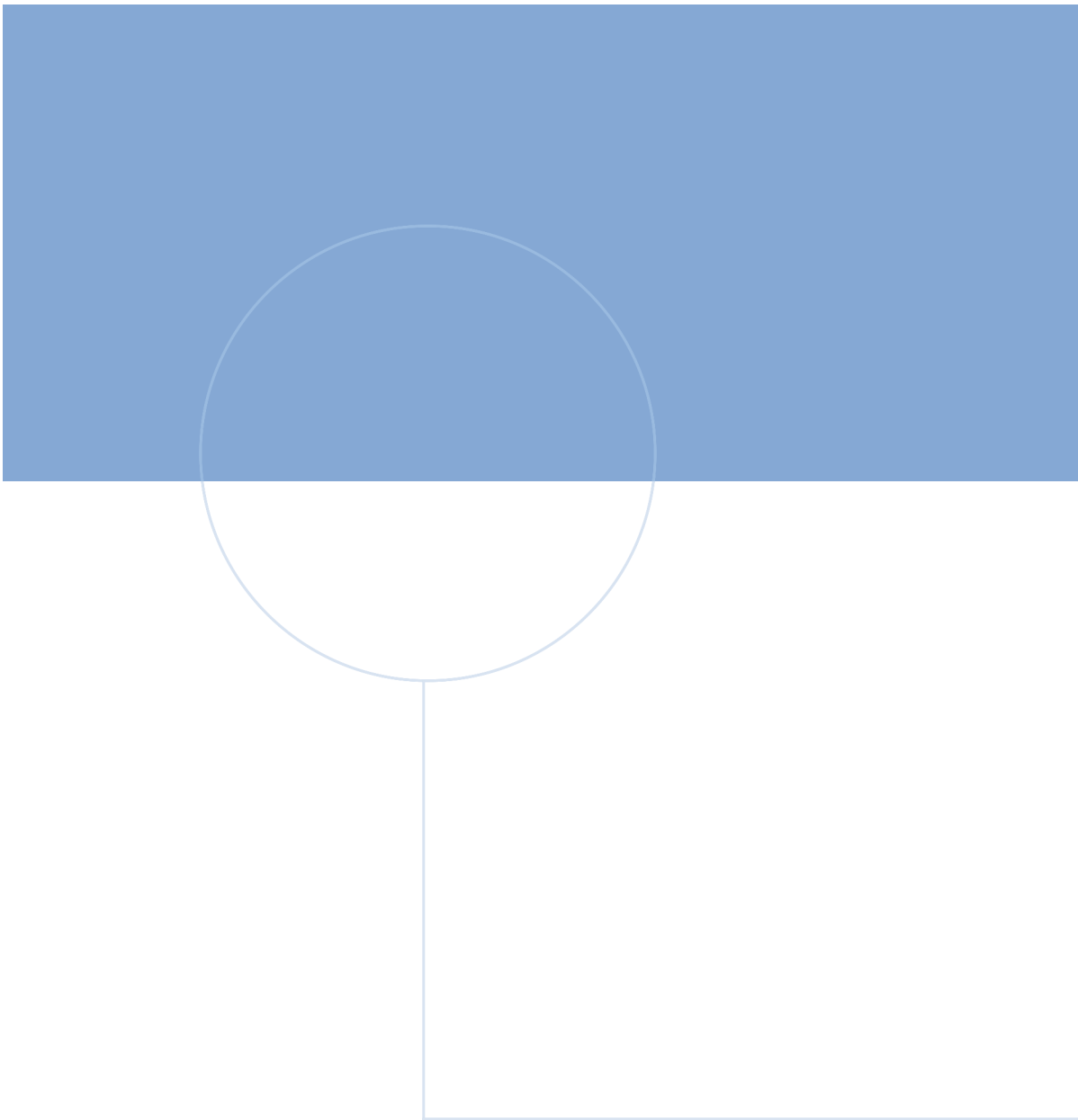
Vallez, C. M., Castro, M., & Contreras, D. (2021). Challenges and opportunities in dock-based bike-sharing rebalancing: a systematic review. *Sustainability*, *13*, 1829. doi:10.3390/su13041829.

Vogel, P. (2016). Service network design of bike sharing systems. In *Service Network Design of Bike Sharing Systems* (pp. 113–135). Springer. doi:10.1007/978-3-319-27735-6.

Voß, S., Fink, A., & Duin, C. (2005). Looking ahead with the pilot method. *Annals of Operations Research*, *136*, 285–302. doi:10.1007/s10479-005-2060-2.

Wang, J., Tsai, C.-H., & Lin, P.-C. (2016). Applying spatial-temporal analysis and retail location theory to public bikes site selection in taipei. *Transportation Research Part A: Policy and Practice*, *94*, 45–61. doi:10.1016/j.tra.2016.08.025.

Zhang, D., Yu, C., Desai, J., Lau, H., & Srivathsan, S. (2017). A time-space network flow approach to dynamic repositioning in bicycle sharing systems. *Transportation research part B: methodological*, *103*, 188–207. doi:10.1016/j.trb.2016.12.006.



NTNU

Norwegian University of
Science and Technology